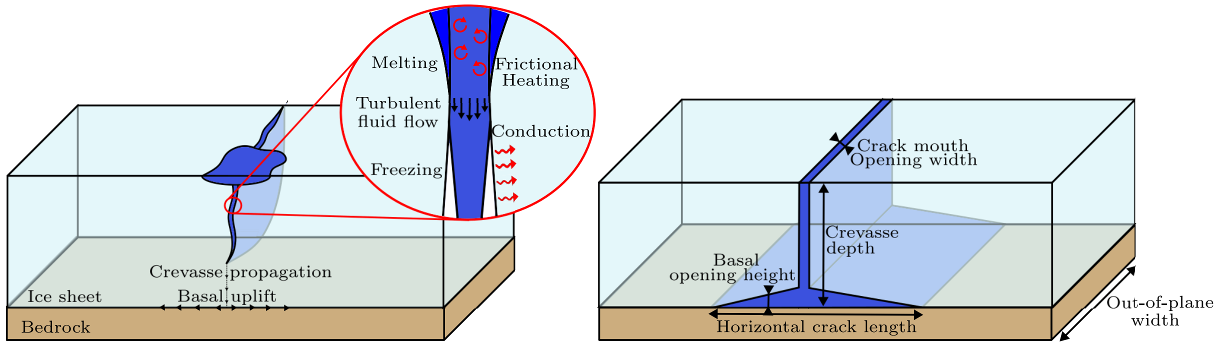

Figure 1: Overview of the system simulated using the presented MATLAB code

IceHydroFrac: A MATLAB code to simulate water-filled crevasse propagation and uplifting of ice sheets

Tim Hageman[a,*], Jessica Mejía[b], Ravindra Duddu[c], Emilio Martínez-Pañeda[a]

[a]*Department of Engineering Science, University of Oxford, Oxford OX1 3PJ, UK*
[b]*Department of Geology, University at Buffalo, Buffalo, NY 14260, USA*
[c]*Department of Civil and Environmental Engineering, Department of Earth and Environmental Sciences, Vanderbilt University, Nashville, TN 37235, USA*

Abstract

Documentation that accompanies the *MATLAB* code IceHydroFrac, available from here. This documentation explains the usage of the implemented finite element framework, and highlight the main files.

If using this module, please cite: T Hageman, JZ Mejia,R Duddu, and E Martinez-Pañeda. *Ice Viscosity Governs Hydraulic Fracture Causing Rapid Drainage of Supraglacial Lakes.* The cryosphere [1].

Keywords: MATLAB, Hydraulic fracture, Greenland Ice Sheet, Crevasse, Numerical simulation, Viscous effects

*Corresponding author
Email address: `tim.hageman@eng.ox.ac.uk` (Tim Hageman)

Contents

1. Introduction

Intro sentences

1.1. Basic usage

All parameters are set within and relevant functions are called from the matlab file “main.m”, and running this file performs the full simulation. Parameters are also set within this file, for instance defining the visco-plastic model through:

main.m

```
        %Interior of ice and rock: Momentum balance
        physics_in{1}.type = "ViscoElastic";
        physics_in{1}.Egroup = "Internal";
        physics_in{1}.young = [9e9; 20e9];       %Youngs modulus of Ice and Rock [Pa]
        physics_in{1}.poisson = [0.33; 0.25];    %Poisson ratio of ice and rock [-]
        physics_in{1}.A0 = 5e-24;                %Glenns law creep coefficient [Pa^-3 s^-1]
        physics_in{1}.Q = 150e3;                 %Energy for scaling creep with temperature
            [J]
        physics_in{1}.TRef = 273.15;             %Reference temperature at which A0 is
            determined [K]
        physics_in{1}.n = 3;                     %Glenns law creep exponent [-]
        physics_in{1}.T_Ice = T_Ice;             %Temperature profile for the ice
```

These parameters are automatically passed along to the relevant physics models once they are initialized. As such, no changes in other files are needed to adapt the simulation set-up for other parameters.

2. Summary of included files

The code is set up in a object-oriented manner, defining matlab classes for each sub-component and providing their accompanying methods. As a result, a clear distinction is made between different components, and each can be used and altered with limited/no impact on other components. Here, the different classes are described. The commenting style employed within the code is compatible with the matlab help function, as such information about all usable methods within a class can be accessed by including the relevant folders, and typing, for instance, “help Solver” to print all variables contained within and all function available from the solver.

2.1. main.m

This is the main file, from which all classes are constructed and the actual simulation is performed. Within it, all properties used within other classes are defined as inputs. These properties are then passed on to initialize the “physics” object, via

main.m

```
        %initialize physics
        physics = Physics(mesh, physics_in, dt0);
```

taking all relevant input parameters within the physics_ in structure, and an initial time step increment dt0. In a similar manner, the mesh is also initialized from a structure of properties,

```
        %load mesh from file
        mesh = Mesh(mesh_in);
        mesh.plot(true, true, false, false);
        mesh.check();
```

which reads the mesh from a file, initializes the required elements and node groups, displays the mesh in a figure, and confirms the mesh is valid and prints statistics related to the area of each separate element group to the output. Finally, the main file performs the time-stepping, calling the function:

```
        %solve current time increment
        solver.Solve();
```

to solve the actual time increments.

After each time increment, outputs are saved into a single structure for later plotting,

```
        %save timedata
        physics.models{8}.updateSurfaceElevation(physics);
        TimeSeries.tvec(end+1) = physics.time;%times of outputs[s]
        TimeSeries.Lfrac(end+1) = mesh.Area(9);%crevasse depth/length
        TimeSeries.Qvec(end+1,:) = physics.models{5}.QMeltTot;%thermal fluxes ('Ice desorbtion', 'Flow produced', 'Melting process')
        TimeSeries.Qinflow(end+1) = physics.models{6}.QTotal(end);%total volume of fluid that has entered the crevasse
        TimeSeries.qCurrent(end+1) = physics.models{6}.qCurrent(end);%current inflow rate
        TimeSeries.upLift(end+1,1) = physics.models{6}.dxCurrent(end);%surface uplift at the centre of the surface
        TimeSeries.upLift(end,2) = physics.models{6}.dyCurrent(end);%surface uplift at the centre of the surface
        TimeSeries.SurfaceDisp(end+1,:,1) = physics.models{6}.surface_dX;%full uplift profile at top surface    (horizontal displacement)
        TimeSeries.SurfaceDisp(end,:,2) = physics.models{6}.surface_dY;%full uplift profile at top surface (vertical displacement)
```

These outputs are:

200. tvec: This contains all time increments at which the other elements within this structure have outputted data. Notably, as the time increment varies between the initialization period ($tvec(i) < 0$) and actual simulations ($tvec(i) \geq 0$) differs, this vector is also required to translate the number of the time step (as used within the naming of full output files) to the actual time of the outputs.
201. Lfrac: This is the length all fractured interfaces. When the crevasse has yet to reach the base, this corresponds to the depth of the crevasse. After reaching the bottom, this is the depth of the crevasse (=the ice thickness) plus the total length of the horizontal cracks (both directions summed together).
202. Qvec: This reports the total thermal energies produced/consumed throughout the simulation. The first element of this vector corresponds to the thermal energy conducted into the ice, the second to the heat produced by the turbulent flow due to friction, and the final element corresponds to the thermal energy used to cause freezing/melting of the crevasse walls. These values are the integrated totals for the complete crack, and are also integrated over the complete time.
203. Qinflow: The total volume of fluid that has entered the crevasse from the inlet at the surface. As with all other outputs, this is given per metre of unit depth.
204. qCurrent: The current fluid inflow at the top inlet, $\partial \mathrm{Qinflow}/\partial t$.
205. upLift: Surface displacement at the centre of the top surface, coinciding with the crevasse. This contains the horizontal displacement (half the crevasse opening height), and the vertical displacement.
207. SurfaceDisp: Vectors containing horizontal and vertical displacements for the complete top surface of the ice-sheet. The coordinates that correspond to the given data points are saved within TimeSeries.SurfaceCoords.

In addition to these time series, full outputs are saved after every 10 time steps,

```
        %save outputs for post-processing and restarting
        if mod(tstep, 10) == 0
            filename = savefolder+string(tstep);
            save(filename, "mesh","physics","solver","t_max","TimeSeries");
        end
```

These output files are appended with the number of the time increment during which the output is saved, and they contain all information required to restart a previously interrupted simulation.

2.2. Models

The models implement parts of the relevant physics. At initialization time, models are constructed by passing their relevant input parameters as:

Models/@ViscoElastic/ViscoElastic.m

```
        function obj = ViscoElastic(mesh, physics, inputs)
```

This also saves pointers to the physics and mesh objects, allowing interacting with these whenever needed. The physics implemented by the model are added within the GetKf function, called whenever an updated tangent stiffness matrix and/or force vector is needed:

```
        function getKf(obj, physics)
```

When models have history-dependent parameters, these are updated by calling the commit function,

```
        function Commit(obj, physics, commit_type)
```

where the commit type indicates either "TimeDep" for committing time-irreversible data (e.g. plastic strains, time-integrated quantities), "irreversibles" for path-irreversible quantities (e.g. cohesive zone models), and "StartIt" for anything that needs to be performed at the start of each time increment.

2.2.1. ViscoElastic.m

This model implements the static momentum balance:

$$-\nabla \cdot \boldsymbol{\sigma} = \mathbf{0} \tag{1}$$

with the weak form of this contribution given by:

$$\int_\Omega \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \boldsymbol{u}^{t+\Delta t} - \boldsymbol{B}^T \boldsymbol{D} \varepsilon_{\mathrm{v}}^{t+\Delta t} \, \mathrm{d}\Omega \tag{2}$$

with the plastic strains determined once per time increment, based on the displacements at the end of the previous time increment:

$$\varepsilon_{\mathrm{v}}^{t+\Delta t} = \varepsilon_{\mathrm{v}}^{t} + \Delta t A \left(\left({\mathbf{u}^t}^T \boldsymbol{B}^T - \varepsilon_{\mathrm{v}}^{t+\Delta t} \right) \boldsymbol{D}^T \boldsymbol{J_2} \boldsymbol{D} \left(\boldsymbol{B} \mathbf{u}^t - \varepsilon_{\mathrm{v}}^{t+\Delta t} \right) \right)^{\frac{n-1}{2}} \boldsymbol{J_2} \boldsymbol{D} \left(\boldsymbol{B} \mathbf{u}^t - \varepsilon_{\mathrm{v}}^{t+\Delta t} \right) \tag{3}$$

with $\boldsymbol{B}$ the displacement to strain mapping matrix, $\epsilon = \boldsymbol{B}\mathbf{u}$, and $\boldsymbol{J}_2$ the deviatoric stress operator:

$$\sigma_{dev} = \boldsymbol{J}\sigma \qquad \boldsymbol{J}_2 = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 \\ -1/3 & -1/3 & 2/3 & 0 \\ -1/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The input parameters for this model are defined by:

main.m

```
        %Interior of ice and rock: Momentum balance
        physics_in{1}.type = "ViscoElastic";
        physics_in{1}.Egroup = "Internal";
        physics_in{1}.young = [9e9; 20e9];       %Youngs modulus of Ice and Rock [Pa]
        physics_in{1}.poisson = [0.33; 0.25];    %Poisson ratio of ice and rock [-]
        physics_in{1}.A0 = 5e-24;                %Glenns law creep coefficient [Pa^-3 s^-1]
        physics_in{1}.Q = 150e3;                 %Energy for scaling creep with temperature [J]
        physics_in{1}.TRef = 273.15;             %Reference temperature at which A0 is determined [K]
        physics_in{1}.n = 3;                     %Glenns law creep exponent [-]
        physics_in{1}.T_Ice = T_Ice;             %Temperature profile for the ice
        physics_in{1}.Hmatswitch = 0;            %Depth of ice-rock interface
```

where “Type” is the name of this model, “Egroup” indicates the domain this model is applied to, and all other are the physical parameters for this model. The temperature of the ice is defined as an interpolation function, which is previously loaded from a file or defined to return a constant by:

```
    %Temperature profile used
    if false
        T_Ice = @(y) -3+0*y;
    else
        load("TProfile.mat","T_Ice");
    end
```

For computational efficiency, Eq. (2) is split into two contributions: The first term contributes to the stiffness matrix and force vector, with the stiffness matrix only needing to be updated when the mesh changes (due to crack propagation):

Models/@ViscoElastic/ViscoElastic.m

```
            if (length(obj.myK) == length(physics.fint))
                recalc = false;
            else
                recalc = true;
            end
```

, with the stiffness matrix assembled in a standard manner as:

```
                        if (obj.Hmatswitch(xy(1,ip),xy(2,ip)) == false)
                            K_el = K_el + B'*obj.D_el2*B*w(ip);
                        else
                            K_el = K_el + B'*obj.D_el*B*w(ip);
                        end
```

where the height of integration points is checked to determine whether elements are located within the ice or rock parts of the domain. This tangent matrix is then used together with a similar force vector resulting for the viscous part of Eq. (2) to add contributions to the global foce vector as:

```
            physics.fint = physics.fint + obj.myK*physics.StateVec ;
            if (obj.A0>0)
                physics.fint = physics.fint + obj.FVisc;
            end
            physics.K = physics.K + obj.myK;
```

As a result, the force vector for the visco-plastic component and the tangent matrix for the linear-elastic components only need to be updated once per time increment, independent of the actual amount of non-linear iterations performed within that increment.

2.2.2. Inertia.m

This model adds inertia terms to the momentum balance, appending the momentum balance from Eq. (1) to read:

$$\rho_\pi \ddot{\mathbf{u}} - \nabla \cdot \sigma = \mathbf{0} \tag{5}$$

where the first term is the newly added term in this model (note: this model is additive to the ViscoElastic.m model, it does not add the stress term by itself). The inputs required to initialize this model are given by:

main.m

```
        %Interior of ice and rock: Contribution due to inertia
        physics_in{2}.type = "Inertia";
        physics_in{2}.Egroup = "Internal";
        physics_in{2}.density = 950;         %Density [kg/m^3]
        physics_in{2}.beta    = 0.4;         %Time discretisation constant (Newmark scheme)
        physics_in{2}.gamma   = 0.75;        %Time discretisation constant (Newmark scheme)
```

where the time discretisation parameters beta and gamma are used to define the acceleration in terms of history parameters (the old velocity and acceleration) and the current displacement as:

$$\dot{\mathbf{u}}^{t+\Delta t} = \frac{\gamma}{\beta \Delta t}\left(\mathbf{u}^{t+\Delta t} - \mathbf{u}^{t}\right) - \left(\frac{\gamma}{\beta} - 1\right)\dot{\mathbf{u}}^{t} - \left(\frac{\Delta t \gamma}{2\beta} - \Delta t\right)\ddot{\mathbf{u}}^{t} \tag{6}$$

$$\ddot{\mathbf{u}}^{t+\Delta t} = \frac{1}{\beta \Delta t^2}\left(\mathbf{u}^{t+\Delta t} - \mathbf{u}^{t}\right) - \frac{1}{\beta \Delta t}\dot{\mathbf{u}}^{t} - \left(\frac{1}{2\beta} - 1\right)\ddot{\mathbf{u}}^{t} \tag{7}$$

Similar to the viscoelastic model, the tangent matrix for this model is calculated once, and then re-used to assemble the global tangent matrix every iteration.

2.2.3. SelfWeight.m

This model adds the gravity contribution to the momentum balance, appending the momentum balance to:

$$\rho_\pi \ddot{\mathbf{u}} - \nabla \cdot \sigma - \rho_\pi \mathbf{g} = \mathbf{0} \tag{8}$$

For simplicity, this gravity force is added to the global force vector, with no differentiation made between internal and external forces. Inputs for this model are the element group the model is acting on, and the densities of ice and rock:

main.m

```
        %Interior of ice and rock: Contribution due to gravity
        physics_in{3}.type = "SelfWeight";
        physics_in{3}.Egroup = "Internal";
        physics_in{3}.density = [910; 2500];    %Density [kg/m^3]
```

2.2.4. FractureCZM.m

This model adds the cohesive fracture model, defined by the interface tractions normal to the fracture surface:

$$\tau_{CZM} = -f_t \mathbf{n} \exp\left(-[\![\mathbf{u}]\!] \cdot \mathbf{n} \frac{f_t}{G_c}\right) \tag{9}$$

It also manages the propagation criteria for fracture propagation, allowing the crack to propagate when $\sigma \cdot \mathbf{n} - f_t > 0$.

To initialize the model, the input parameters required are:

main.m

```
        %Fracture interface: Cohesive zone model and propagation
        physics_in{4}.type = "FractureCZM";
        physics_in{4}.Egroup = "Fracture";
        physics_in{4}.energy = 10;          %Fracture release energy [J/m^2]
        physics_in{4}.dummy = 0*1e10;       %Dummy stiffness to prevent walls from
            penetrating
        physics_in{4}.Hmatswitch = 0;       %Depth of ice-rock interface
        physics_in{4}.T_ice = T_Ice;        %Temperature profile of ice
```

where the temperature profile that is given as input is used to obtain the tensile strength via:

Models/@FractureCZM/FractureCZM.m

```
            obj.ft = @(T) 2.0-0.0068*(T+273.15);
```

For the surface tractions, the maximum displacement is saved as a history variable, saved during the assembly of the force and tangent stiffness matrix:

```
                        if (h>=hstOld)
                            hloc = h;
                            tau(1) = f_t * exp(-f_t*h/obj.energy);
                            dtaudh(1,1) = -f_t.^2/obj.energy * exp(-f_t*h/obj.energy);
```

```
                elseif (h>0)
                     hloc = hstOld;
                     tau(1) = f_t * exp(-f_t*hstOld/obj.energy)*h/hstOld;
                     dtaudh(1,1) = f_t * exp(-f_t*hstOld/obj.energy)/hstOld;
                else
                    hloc = hstOld;
                    %tractions via no-pen condition
                end
                newHist(n_el, ip) = hloc;
```

which also produces the surface traction normal to the surface, tau(1), and its derivative dtaudh(1,1). The tangent component of this traction is set to zero, and during assembly this vector is rotated from the local to the global coordinate system. To improve the stability of the no-penetration condition, used to enforce contact between the crevasse walls, a lumped integration scheme is used, with sets of nodes contributing on a set-by-set basis to the overall force vector as:

```
            for cp=1:length(C_Lumped)
                NL = zeros(size(N, 2), 1);
                NL(cp) = 1.0;
                Nd = obj.getNd(NL);

                n_est = nvec(1,:);
                if (n_est*Nd*XY < 0)
                    f_el = f_el + C_Lumped(cp)*obj.dummy*(Nd'*(n_est'*n_est)*Nd)*XY;
                    K_el = K_el + C_Lumped(cp)*obj.dummy*(Nd'*(n_est'*n_est)*Nd);
                end
            end
```

with this condition only activating when the crevasse opening height is negative.

The fracture criterion is evaluated in the function

```
        function [fc, dofsXY] = get_fc(obj, physics)
```

This function queries the elements ahead of the crack tip to obtain its stresses

```
            [direction, elem, ips] = obj.mesh.getNextFracture();
            stresses = physics.Request_Info("stresses",elem,"Interior");
```

which are compared to the temperature-dependent tensile strength. If the propagation criterion is exceeded, the mesh is addapted to include a newly inserted interface element:

```
            [fc, ~] = obj.get_fc(physics);
            if (fc>=0 && physics.time>=0)
                obj.mesh.Propagate_Disc_New(physics);
                Irr = true;
            end
```

2.2.5. FractureFluid.m

FractureFluid.m implements all the fluid flow and thermal related equations for the water within the crevasse, and is defined by the input parameters:

main.m

```
        %Fracture interface: Fluid flow and melting
        physics_in{5}.type = "FractureFluid";
        physics_in{5}.Egroup = "Fracture";
        physics_in{5}.visc = 1.0e-3;          %water viscosity [Pa s]
        physics_in{5}.Kf   = 1.0e9;           %Water compressibility [Pa]
        physics_in{5}.FlowModel = "FrictionFactor";   %"CubicLaw";"FrictionFactor", Model used for fluid flow within crevasse
        physics_in{5}.melt = true;            %Include wall melting
        physics_in{5}.freeze = true;          %Include wall freezing
        physics_in{5}.rho_ice = 910;          %Density of ice [kg/m^3]
        physics_in{5}.rho_water = 1000;       %Density of water [kg/m^3]
```

```
        physics_in{5}.cp_ice = 2115;          %heat capacity of ice [J/kg]
        physics_in{5}.melt_heat = 335000;     %latent heat of ice-water tansition [J/kg]
        physics_in{5}.T_ice = T_Ice;          %Temperature profile
        physics_in{5}.k_ice = 2;              %Thermal conductivity of ice [J/K/m^2]
```

Notably, the input "FlowModel" allows selecting either turbulent flow, based on a friction factor approach via:

$$q = -2\rho_{\mathrm{w}}^{-\frac{1}{2}} k_{\mathrm{wall}}^{-\frac{1}{6}} f_0^{-\frac{1}{2}} h^{\frac{5}{3}} \left| \frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right|^{-\frac{1}{2}} \left(\frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right) \tag{10}$$

or laminar flow, using the analytic solution for uni-directional pressure-driven fluid flow between flat plates:

$$q = \frac{h^3}{12\mu} \left(\frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right) \tag{11}$$

with this choice of model being set as either "FrictionFactor" for turbulent flow, or "CubicLaw" for laminar flow. This choice is applied to the complete crevasse, and no checking is performed whether the fluid velocity warrants laminar or turbulent flow (based on simulations performed for our paper, the Reynolds number indicates turbulent for our use cases).

This fluid flux is used within the mass balance:

$$\frac{\partial q}{\partial \xi} + \dot{h} - \frac{\rho_{\mathrm{i}}}{\rho_{\mathrm{w}}} \dot{h}_{\mathrm{melt}} + \frac{h}{K_{\mathrm{w}}} \dot{p} = 0 \tag{12}$$

which also requires changes in local opening height, and melting rate. These are obtained by solving the coupled system of equations within each integration point:

$$q = -2\rho_{\mathrm{w}}^{-\frac{1}{2}} k_{\mathrm{wall}}^{-\frac{1}{6}} f_0^{-\frac{1}{2}} h^{\frac{5}{3}} \left| \frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right|^{-\frac{1}{2}} \left(\frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right) \tag{13}$$

$$0 = \rho_{\mathrm{i}} \mathcal{L} \dot{h}_{\mathrm{melt}} + \frac{k^{\frac{1}{2}} T_\infty \rho_{\mathrm{i}}^{\frac{1}{2}} c_{\mathrm{p}}^{\frac{1}{2}}}{\pi^{\frac{1}{2}} (t - t_0)^{\frac{3}{2}}} + q \left(\frac{\partial p}{\partial \xi} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right) \tag{14}$$

$$h = h_{\mathrm{melt}} + \mathbf{n} \cdot [\![\mathbf{u}]\!] \tag{15}$$

Defining the momentum balance of the fluid, thermal balance, and total opening height respectively. This system is solved in a coupled manner, using:

$$\boldsymbol{C}_{ip} \begin{bmatrix} \mathrm{d}q_{ip}^{t+\Delta t} \\ \mathrm{d}h_{\mathrm{melt}\,ip}^{\;t+\Delta t} \\ \mathrm{d}h_{ip}^{t+\Delta t} \end{bmatrix}_{i+1} = - \begin{bmatrix} f_{1,ip} \\ f_{2,ip} \\ f_{3,ip} \end{bmatrix}_i , \tag{16}$$

where the integration-point level tangent matrix given by:

$$\boldsymbol{C}_{ip} = \begin{bmatrix} 1 & 0 & \frac{10}{3} \rho_{\mathrm{w}}^{-\frac{1}{2}} k_{\mathrm{wall}}^{-\frac{1}{6}} f_0^{-\frac{1}{2}} h_{ip}^{t+\Delta t\, \frac{2}{3}} \left| \boldsymbol{\nabla} \mathbf{N}_{\mathrm{f}} \mathbf{p}^{t+\Delta t} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right|^{-\frac{1}{2}} \left(\boldsymbol{\nabla} \mathbf{N}_{\mathrm{f}} \mathbf{p}^{t+\Delta t} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right) \\ \boldsymbol{\nabla} \mathbf{N}_f \mathbf{p}^{t+\Delta t} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} & \frac{\rho_{\mathrm{i}} \mathcal{L}}{\Delta t} & 0 \\ 0 & -1 & 1 \end{bmatrix} \tag{17}$$

and during the assembly of the tangent system matrix for the global system, consistent tangent matrices are obtained via:

$$\begin{bmatrix} \frac{\partial q}{\partial \mathbf{p}} & \frac{\partial q}{\partial \mathbf{u}} \\ \frac{\partial h_{\mathrm{melt}}}{\partial \mathbf{p}} & \frac{\partial h_{\mathrm{melt}}}{\partial \mathbf{u}} \\ \frac{\partial h}{\partial \mathbf{p}} & \frac{\partial h}{\partial \mathbf{u}} \end{bmatrix} = -\boldsymbol{C}_{ip}^{-1} \begin{bmatrix} \rho_{\mathrm{w}}^{-\frac{1}{2}} k_{\mathrm{wall}}^{-\frac{1}{6}} f_0^{-\frac{1}{2}} h_{ip}^{t+\Delta t\, \frac{5}{3}} \left| \boldsymbol{\nabla} \mathbf{N}_{\mathrm{f}} \mathbf{p}^{t+\Delta t} - \rho_{\mathrm{w}} \mathbf{g} \cdot \mathbf{s} \right|^{-\frac{1}{2}} \boldsymbol{\nabla} \mathbf{N}_{\mathrm{f}} & 0 \\ q_{ip}^{t+\Delta t} \boldsymbol{\nabla} \mathbf{N}_{\mathrm{f}} & 0 \\ 0 & -\boldsymbol{N}_{\mathrm{d}} \end{bmatrix} \tag{18}$$

Within the code, this procedure is implemented through

Models/@FractureFluid/FractureFluid.m

```
        function [qx, hmelt, h, derivs, Qprod] = get_qx_hmelt_h(obj, dp_dx, u, tfr, tOld,
            dt, hMelt_hist, initGuess, Ice_temp)
```

which takes the pressure gradient and displacement jump (and saved history variables), and returns the fluid flux, melting height, and total opening height and their derivatives. This function is aided by

```
        function [f, C, D, Qres] = getFracK(obj, sol, dp_dx, u, tfr, tOld, dt, hMelt_hist,
            Ice_temp)
```

which returns the C and D matrices defined above, and the force vector. Additionally, this uses the function

```
        function [qxFlow, dqx_dh, dqx_dpdx] = getFlow(obj, dp_dx, h)
            if (h<0)
                h=0;
            end
            if (obj.FlowModel == "CubicLaw")
                qxFlow = -h^3/(12*obj.visc)*dp_dx;
                dqx_dh = -3*(h)^2/(12*obj.visc)*dp_dx;
                dqx_dpdx = -(h)^3/(12*obj.visc);
            end
            if (obj.FlowModel == "FrictionFactor")
                k = 1e-2; f0 = 0.143;
                preFac = obj.rho_water^(-0.5)*k^(-1/6)*sqrt(4)*f0^(-0.5);
                qxFlow   = -preFac    *(max(1,abs(dp_dx)))^(-0.5)*dp_dx*h^(5/3);
                dqx_dh   = -preFac*5/3*(max(1,abs(dp_dx)))^(-0.5)*dp_dx*h^(2/3);
                dqx_dpdx = -preFac*0.5*(max(1,abs(dp_dx)))^(-0.5)*h^(5/3);
            end
        end
```

to determine the fluid flux, depending on the used model.

2.2.6. LakeBoundary.m

2.2.7. Constrainer.m

2.3. Mesh

3. Sample results

References

[1] T. Hageman, J. Z. Mejia, R. Duddu, E. Martinez-Pañeda, Ice viscosity governs hydraulic fracture causing rapid drainage of supraglacial lakes [Submitted], The Cryosphere.