Stokes problems using Firedrake

A glacier and ice sheet tutorial

Ed Bueler

University of Alaska Fairbanks

April 2021

github.com/bueler/stokes-ice-tutorial

Outline

`github.com/bueler/stokes-ice-tutorial`

1. the problem and the equations
2. `stage1/` linear Stokes
3. `stage2/` Glen-Stokes
4. `stage3/` extruded meshes
5. `stage4/` bells and whistles
6. `stage5/` 3D glaciers in parallel

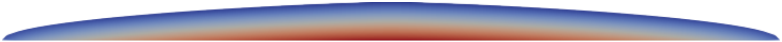
exclamation points

- I cannot explain everything in an hour!
- ask questions!
- please try the codes!
- feel free to send future questions by email!

the glacier dynamics problem

- conservation principles apply to ice sheets; each one is a PDE
  - mass ✓ today
  - momentum ✓ today
  - energy ✗ *not today*
- Stokes model $=$ momentum conservation $+$ incompressibility
  - incompressibility is one aspect of mass conservation
  - no attempt *today* to model geometry changes using surface balance, the other aspect
- the Stokes model determines velocity and pressure from given geometry

velocity


pressure


$$
\begin{aligned}
-\nabla \cdot \tau + \nabla p &= \rho_{\text{i}} \mathbf{g} && \textit{stress balance} \\
\nabla \cdot \mathbf{u} &= 0 && \textit{incompressibility} \\
\tau &= B_n |D\mathbf{u}|^{(1/n)-1} D\mathbf{u} && \textit{Glen flow law}
\end{aligned}
$$

John Glen

- $\mathbf{u}$ is velocity, p is pressure, and τ is the deviatoric stress tensor
- constants: $\rho_{\text{i}} = 910\,\text{kg}\,\text{m}^{-3}$, $g = 9.81\,\text{m}\,\text{s}^{-2}$, $n = 3$, $B_n = 6.8 \times 10^7\,\text{Pa}\,\text{s}^{1/3}$
- viscosity regularization with $\epsilon = 10^{-4}$ and $D_0 = 1\,\text{a}^{-1}$:

$$
\nu_\epsilon(|D\mathbf{u}|) = \frac{1}{2} B_n \left(|D\mathbf{u}|^2 + \epsilon\, D_0^2\right)^{((1/n)-1)/2}
$$

- eliminate τ to give system for $\mathbf{u}, p$:

$$
\begin{aligned}
-\nabla \cdot \left(2\nu_\epsilon(|D\mathbf{u}|)\, D\mathbf{u}\right) + \nabla p &= \rho_{\text{i}} \mathbf{g} && \textit{momentum conservation} \\
\nabla \cdot \mathbf{u} &= 0 && \textit{(bulk) mass conservation}
\end{aligned}
$$

boundary conditions, as simple as possible

- stress-free top:

$$\sigma\mathbf{n} = \left(2\nu_\epsilon(|D\mathbf{u}|)D\mathbf{u} - pI\right)\mathbf{n} = \mathbf{0}$$

- no slip base:

$$\mathbf{u} = \mathbf{0}$$

  - no attempt *today* to model sliding

the linear Stokes equations

George Stokes

- if we make viscosity constant (ν_0) then we get the linear Stokes system:

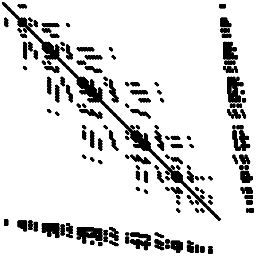
$$\begin{aligned} -\nabla \cdot (2\nu_0\, D\mathbf{u}) + \nabla p &= \rho_{\text{i}} \mathbf{g} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned}$$

- with reformatting:

$$\begin{aligned} -\nu_0 \nabla^2 \mathbf{u} + \nabla p &= \rho_{\text{i}} \mathbf{g} \\ -\nabla \cdot \mathbf{u} \qquad\quad &= 0 \end{aligned}$$

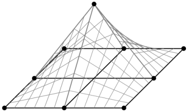
- it has symmetric block structure

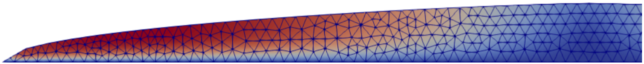
$$\begin{bmatrix} A & B^\top \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}$$



finite elements . . . with no details

- exact solution of the Stokes model is impossible for most geometries
  - slab-on-a-slope is an exception
- so we solve numerically using the finite element (FE) method
  - the whole point of Firedrake is that we *don't* need to know the details of FE
- in summary, the FE method allow us to start from a mesh of triangles (or quadrilaterals, prisms, . . .) over our domain, then express the approximate solution in terms of functions built from the mesh, then turn the weak form of the equations into a finite system by using test functions built from the mesh, and then solve those equations somehow, and then there is a lot of fine print lorem ipsum dolor sit amet consectetur adipiscing elit etiam ex neque rhoncus a nunc tristique dignissim euismod magna fusce pharetra tempor sapien vitae scelerisque duis in quam rhoncus suscipit leo quis pellentesque diam suspendisse mollis nisi et eros ornare tincidunt maecenas sed pellentesque ex id mattis libero morbi erat orci fermentum eu dui ac . . .





- we need to write a weak form!

weak form

- start from the original equations, also called the *strong form*:

$$-\nabla \cdot (2\nu_\epsilon(|D\mathbf{u}|)\, D\mathbf{u}) + \nabla p = \rho_\mathrm{i}\mathbf{g} \qquad (1)$$
$$\nabla \cdot \mathbf{u} = 0 \qquad (2)$$

- multiply (1) by test velocity $\mathbf{v}$ and (2) by test pressure q, integrate by parts, and combine into one nonlinear residual function:

$$F(\mathbf{u}, p; \mathbf{v}, q) = \int_\Omega 2\nu_\epsilon(|D\mathbf{u}|)\, D\mathbf{u} \;:\; D\mathbf{v} - p\nabla \cdot \mathbf{v} - q\nabla \cdot \mathbf{u} - \rho_\mathrm{i}\mathbf{g} \cdot \mathbf{v}\, dx$$

- in Firedrake's language (= *Unified Form Language*) it looks like:

```
fbody = Constant((0.0, 0.0, - rho * g))
Du2 = 0.5 * inner(D(u), D(u)) + (eps * Dtyp)**2.0
nu = 0.5 * Bn * Du2**((1.0 / n - 1.0)/2.0)
F = ( inner(2.0 * nu * D(u), D(v)) \
      - p * div(v) - q * div(u) - inner(fbody, v) ) * dx
```

- the actual weak form "equation" is the statement

$$F(\mathbf{u}, p; \mathbf{v}, q) = 0 \qquad \text{for all } \mathbf{v} \text{ and } q$$

mixed finite elements for fluid problems

- the other detail we need to know about in practice:

  choose separate function spaces for the velocity and the pressure

  and

  choose spaces that the experts say are "stable"

- Firedrake makes this elegant:

```
V = VectorFunctionSpace(mesh, 'Lagrange', 2)    # u space
W = FunctionSpace(mesh, 'Lagrange', 1)          # p space
Z = V * W
up = Function(Z)
u, p = split(up)
v, q = TestFunctions(Z)
```

running Firedrake programs

- now I'm actually going to run some Python Firedrake code!
- install Firedrake following these directions:

  `firedrakeproject.org/download.html`

- activate the Python virtual environment every time you use Firedrake:

```
$ unset PETSC_DIR; unset PETSC_ARCH;  # may be needed
$ source ~/firedrake/bin/activate
```

  - I use an alias `drakeme` for this

- other tools I will need:
  - Gmsh `gmsh.info`
  - Paraview `paraview.org`

- also grab my tutorial (codes and slides):

```
git clone https://github.com/bueler/stokes-ice-tutorial.git
```

- *purpose:* solve linear Stokes on a trapezoidal glacier
  - simplified weak form

$$F(\mathbf{u},p;\mathbf{v},q) = \int_\Omega 2\nu_0\, D\mathbf{u} : D\mathbf{v} - p\nabla\cdot\mathbf{v} - q\nabla\cdot\mathbf{u} - \rho_i \mathbf{g}\cdot\mathbf{v}\,dx$$

- *source files:* `domain.geo`, `solve.py` ← inspect these!
- *generated files:* `domain.msh`, `domain_0.vtu`, `domain.pvd`

```
$ cd stage1/
$ gmsh -2 domain.geo                    # mesh the domain
$ gmsh domain.msh                       # view the mesh
$ ./solve.py                            # solve linear Stokes
$ paraview domain.pvd                   # view the solution
```

speed $|\mathbf{u}|$

Newton's method

- linear Stokes (`stage1/`) needs only a single linear system:

$$\begin{bmatrix} A & B^\top \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}$$

  - Firedrake asks PETSc's KSP component to solve this
- but for Glen-Stokes the weak form is nonlinear in **u**:

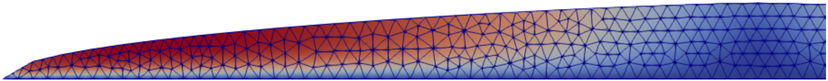
$$F(\mathbf{u}, p; \mathbf{v}, q) = \int_\Omega 2\, \nu_\epsilon(|D\mathbf{u}|)\, D\mathbf{u} : D\mathbf{v} - p \nabla\cdot\mathbf{v} - q\nabla\cdot\mathbf{u} - \rho_i \mathbf{g}\cdot\mathbf{v}\, dx$$

- Firedrake automatically applies Newton's method
  - because we write `solve(F == 0,...`
- all you need to know about Newton's iteration:
  - it is repeated linearization around the current iterate
  - Firedrake uses UFL symbolic differentiation for linearization
  - Firedrake asks PETSc's SNES to do the Newton iteration
    - SNES options will monitor and control the Newton iteration

Isaac Newton

- *purpose:* solve Glen-Stokes on a flat-bed glacier
- *source files:* `domain.py`, `solve.py` ← inspect these!
- *generated files:* `dome.geo`, `dome.msh`, `dome_0.vtu`, `dome.pvd`

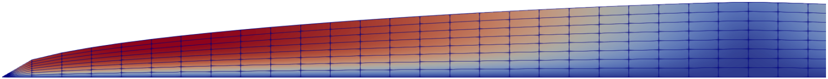
```
$ cd stage2/
$ ./domain.py                      # generate geometry
$ gmsh -2 dome.geo                 # mesh domain
$ gmsh dome.msh                    # view mesh
$ ./solve.py -s_snes_monitor -s_snes_converged_reason
$ paraview dome.pvd                # view solution
```



speed $|\mathbf{u}|$

- *purpose:* solve same problem using an extruded quadrilateral mesh
- *source files:* `solve.py` ← inspect this!
- *generated files:* `dome_0.vtu`, `dome.pvd`

```
$ cd stage3/
$ ./solve.py -s_snes_monitor -s_snes_converged_reason
$ paraview dome.pvd
```



speed $|\mathbf{u}|$

convergence?

- `stage3/` code `solve.py` allows adjustable resolution, e.g.:
  ```
  $ ./solve.py -mx 40 -mz 4
  ```
- velocity results are reasonable and consistent:

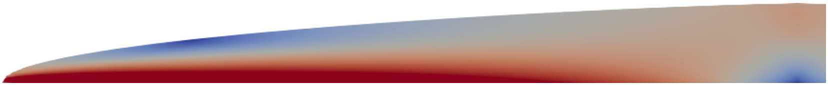
mesh	$\Delta x \times \Delta z$ (m)	av. $\lvert\mathbf{u}\rvert$ (m/a)	max. $\lvert\mathbf{u}\rvert$ (m/a)
20×2	1000×500	1813	3512
40×4	500×250	1787	3293
80×8	250×125	1769	3223
160×16	125×63	1762	3199
320×32	63×31	1759	3191
640×64	31×16	1758	3190

  - *unfortunately*, the finest resolution needed $\epsilon = 10^{-2}$ to get “unstuck” from the $\mathbf{u}, p = \mathbf{0}, 0$ initial iterate
- we need better initial iterates for high-resolution runs

`stage4/` bells and whistles

- *purpose:* additional robust and/or useful features
  - rescale the equations — *better conditioning*
  - vertical grid sequencing — *better initial iterates*
  - 100ϵ on coarse meshes in sequencing — *better initial iterates*
  - generate stress tensor τ from solution — *diagnostic*
  - generate effective viscosity ν_ϵ from solution — *diagnostic*
- *source files:* `solve.py` ← inspect this!
- *generated files:* `dome_0.vtu`, `dome.pvd`

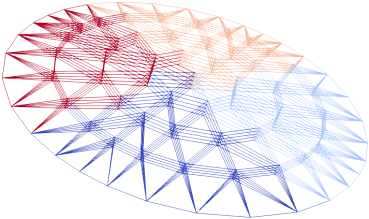
```
$ cd stage4/
$ ./solve.py
$ ./solve.py -mx 320 -mz 2 -refine 2 \
    -s_snes_atol 1.0e-2 -s_snes_monitor
$ paraview dome.pvd
```



deviatoric stress magnitude $||\tau||$

- *purpose:* 3D ice sheet in parallel on bumpy bed
- *source files:* `solve.py` ← inspect this!
- *generated files:* `dome_0_`*k*`.vtu`, `dome_0.pvtu`, `dome.pvd`

```
$ cd stage5/
$ mpiexec -n 4 ./solve.py -refine 0 \
    -s_snes_atol 1.0e-2 -s_snes_monitor
$ paraview dome.pvd
$ mpiexec -n 12 ./solve.py -bumpy -baserefine 5 \
    -refine 2 -s_snes_atol 1.0e-2 -s_snes_monitor \
    -o hires.pvd
$ paraview hires.pvd
```



github.com/bueler/stokes-ice-tutorial

- Firedrake: `firedrakeproject.org`
  - tutorials & manual: `.../documentation.html`
  - Jupyter notebooks page: `.../notebooks.html`
- PETSc website: `www.mcs.anl.gov/petsc`
- for Glen-Stokes eqns, see Ch. 1 by Hewitt:

  Fowler & Ng, ed., *Glaciers and Ice Sheets in the Climate System: The Karthaus Summer School Lecture Notes*, Springer 2021
- for finite elements and linear Stokes:

  Elman, Silvester, & Wathen, *Finite Elements and Fast Iterative Solvers, With Applications in Incompressible Fluid Dynamics*, Oxford 2014, 2nd ed.
- for PETSc, Firedrake, and Stokes (Ch. 14):

  Bueler, *PETSc for Partial Differential Equations: Numerical Solutions in C and Python*, SIAM 2021