

# Stokes problems using Firedrake

A glacier and ice sheet tutorial

Ed Bueler

University of Alaska Fairbanks

version June 2023

[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

# exclamation points

- I cannot explain everything in an hour!
- ask questions!
- please try the codes!
- send future questions!
  - by email to [elbueler@alaska.edu](mailto:elbueler@alaska.edu), or
  - using [github issues](#)

[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      linear Stokes

stage2/      Glen-Stokes

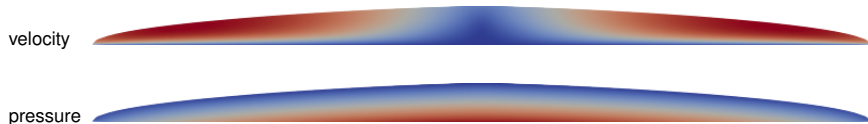
stage3/      extruded meshes

stage4/      bells and whistles

stage5/      3D glaciers in parallel

# the glacier dynamics problem, without evolution of geometry

- conservation principles apply to ice sheets:
  - 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 of mass conservation
- the Stokes model determines velocity and pressure from given geometry



# Glen-Stokes equations

$$-\nabla \cdot \tau + \nabla p = \rho_i \mathbf{g}$$

*stress balance*

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

*incompressibility*

$$\tau = B_n |\mathbf{Du}|^{(1/n)-1} \mathbf{Du}$$

*Glen flow law*



John Glen

- $\mathbf{u}$  is velocity,  $p$  is pressure, and  $\tau$  is the deviatoric stress tensor
- constants:  $\rho_i = 910 \text{ kg m}^{-3}$ ,  $g = 9.81 \text{ m s}^{-2}$ ,  $n = 3$ ,  $B_n = 6.8 \times 10^7 \text{ Pa s}^{1/3}$ 
  - isothermal, so  $B_n$  is constant
- now regularize viscosity using  $\epsilon = 10^{-4}$  and  $D_0 = 1 \text{ a}^{-1}$ :

$$\nu_\epsilon(|\mathbf{Du}|) = \frac{1}{2} B_n (|\mathbf{Du}|^2 + \epsilon D_0^2)^{((1/n)-1)/2}$$

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

$$-\nabla \cdot (2\nu_\epsilon(|\mathbf{Du}|) \mathbf{Du}) + \nabla p = \rho_i \mathbf{g}$$

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

## boundary conditions, as simple as possible

- stress-free top:

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



- no slip base:

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

- no attempt *today* to model sliding

# the linear Stokes equations

- if we make viscosity constant ( $\nu_0$ ) then we get a linear Stokes system:

$$\begin{aligned}-\nabla \cdot (2\nu_0 D\mathbf{u}) + \nabla p &= \rho_1 \mathbf{g} \\ \nabla \cdot \mathbf{u} &= 0\end{aligned}$$

- with reformatting, using  $\nabla \cdot D\mathbf{u} = \nabla^2 \mathbf{u}$ :

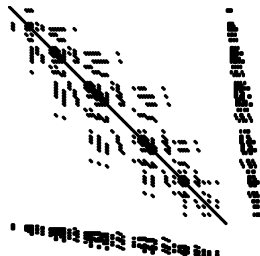
$$\begin{aligned}-\nu_0 \nabla^2 \mathbf{u} + \nabla p &= \rho_1 \mathbf{g} \\ -\nabla \cdot \mathbf{u} &= 0\end{aligned}$$

- which has symmetric block structure

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

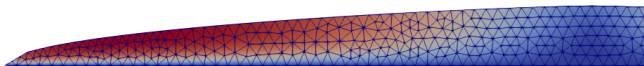
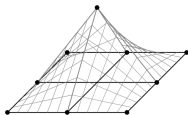


George Stokes



## finite elements ... with no details

- exact solution of the Stokes model is impossible for most geometries
  - *exercise.* slab-on-a-slope is an exception
- so we solve numerically using the finite element (FE) method
- 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 more and more details which I am skipping ...



- the whole point of Firedrake is to *not* see the details of FE
- one needs to know: the first step is to write a weak form



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

$$-\nabla \cdot (2\nu_\epsilon(|D\mathbf{u}|) D\mathbf{u}) + \nabla p = \rho_i \mathbf{g} \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (2)$$

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

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

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

```
fbody = rho * Constant((0.0, 0.0, - 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 equation solved by the FE method is the statement

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

- the other details we need to know about:
  1. choose separate function spaces for the velocity and the pressure
  2. choose spaces that the experts say are “stable”
- Firedrake makes choosing  $P_2 \times P_1$  (Taylor-Hood) elements very easy:

```
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)
```

- other mixed spaces are just as easy

[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      **linear Stokes**

stage2/      Glen-Stokes

stage3/      extruded meshes

stage4/      bells and whistles

stage5/      3D glaciers in parallel

# running Firedrake programs

- now I'm actually going to run some code
- grab my tutorial (codes and these slides):

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

- install Firedrake following these directions:

[firedrakeproject.org/download.html](https://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](https://gmsh.info)
  - Paraview [paraview.org](https://paraview.org)

- *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
```



[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      linear Stokes

stage2/      **Glen-Stokes**

stage3/      extruded meshes

stage4/      bells and whistles

stage5/      3D glaciers in parallel

# 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  $\mathbf{u}$ :

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

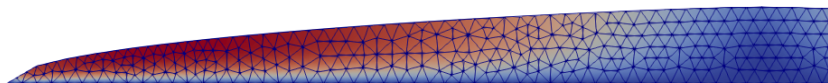
- Firedrake automatically applies Newton's method if we write the problem as `solve(F == 0, ...)`
- all you need to know about Newton's iteration:
  - it does linearization around each iterate
  - Firedrake uses UFL symbolic differentiation for linearization
  - Firedrake asks PETSc's SNES to do the 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  
$ gmsh -2 dome.geo  
$ gmsh dome.msh  
$ ./solve.py -s_snes_monitor -s_snes_converged_reason  
$ paraview dome.pvd
```



speed  $|\mathbf{u}|$



[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      linear Stokes

stage2/      Glen-Stokes

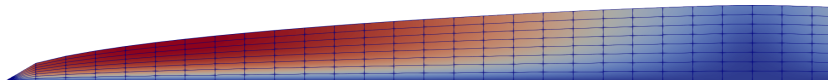
stage3/      **extruded meshes**

stage4/      bells and whistles

stage5/      3D glaciers in parallel

- *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/ `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. $ \mathbf{u} $ (m/a)	max. $ \mathbf{u} $ (m/a)
$40 \times 4$	$500 \times 250$	1787	3293
$80 \times 8$	$250 \times 125$	1769	3223
$160 \times 16$	$125 \times 63$	1762	3199
$320 \times 32$	$63 \times 31$	1759	3192
$640 \times 64$	$31 \times 16$	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

[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      linear Stokes

stage2/      Glen-Stokes

stage3/      extruded meshes

stage4/      **bells and whistles**

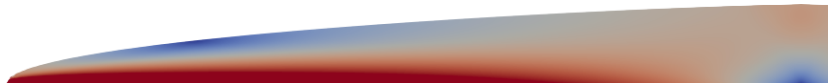
stage5/      3D glaciers in parallel

- *purpose*: additional robust and/or useful features
  - rescale the equations
  - vertical grid sequencing
  - 100€ on coarse meshes in sequencing
  - generate stress tensor  $\tau$  from solution
  - generate effective viscosity  $\nu_\epsilon$  from solution
- *source files*: `solve.py`
- *generated files*: `dome_0.vtu`, `dome.pvd`

*better conditioning*  
*better initial iterates*  
*better initial iterates*  
*diagnostic*  
*diagnostic*

← inspect this!

```
$ 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  $\|\tau\|$

[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

stage 0      the problem and the equations

stage1/      linear Stokes

stage2/      Glen-Stokes

stage3/      extruded meshes

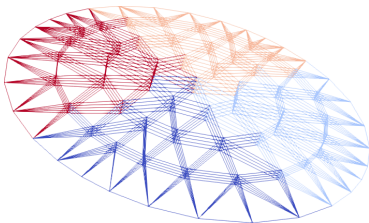
stage4/      bells and whistles

stage5/      3D glaciers in parallel

- *purpose*: 3D ice sheet in parallel on bumpy bed
- *source files*: `solve.py`
- *generated files*: `dome_0_k.vtu`, `dome_0.pvtu`, `dome.pvd`

← inspect this!

```
$ cd stage5/  
$ mpiexec -n 4 ./solve.py \  
    -s_snes_atol 1.0e-2 -s_snes_monitor  
$ paraview dome.pvd  
$ mpiexec -n 4 ./solve.py -refine 2 -baserefine 3 \  
    -s_snes_atol 1.0e-2 -s_snes_monitor -o finer.pvd  
$ paraview finer.pvd
```



# solver performance limitations

- the run below is as far as I can refine on my big desktop
  - 100 GB ram, plenty of cores
  - 38 minutes run time
- *limiting factor*: the direct solver for each Newton step linear system uses too much memory when generating the LU factors
  - direct solvers experience *fill-in*, especially in 3D, and they are slow
  - E. Bueler (2022). *Performance analysis of high-resolution ice-sheet simulations*, J. Glaciol., [10.1017/jog.2022.113](https://doi.org/10.1017/jog.2022.113)
- for higher resolution we need a scalable solver, such as a solver using Schur-complement preconditioning and multigrid on the **u-u** block
  - T. Isaac, G. Stadler, & O. Ghattas (2015). *Solution of nonlinear Stokes equations discretized by high-order finite elements on nonconforming and anisotropic meshes, with application to ice sheet dynamics*, SIAM J. Sci. Comput. 37 (6), B804–B833, [10.1137/140974407](https://doi.org/10.1137/140974407)
- another talk would be needed to discuss and demonstrate a better solver

```
$ mpiexec -n 12 ./solve.py -refine 2 -baserefine 4 \  
-s_snes_atol 1.0e-2 -s_snes_monitor -o hires.pvd  
$ paraview hires.pvd
```



[github.com/bueler/stokes-ice-tutorial](https://github.com/bueler/stokes-ice-tutorial)

- Firedrake: [firedrakeproject.org](https://firedrakeproject.org)
  - tutorials & manual: [.../documentation.html](https://firedrakeproject.org/documentation.html)
  - Jupyter notebooks page: [.../notebooks.html](https://firedrakeproject.org/notebooks.html)
- PETSc website: [www.mcs.anl.gov/petsc](https://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



*Firedrake*

 PETSc

