Solving PDEs with Firedrake: hyperelasticity

Patrick E. Farrell

University of Oxford

February 2024

We will build a solver for a nontrivial problem: compressible hyperelasticity.

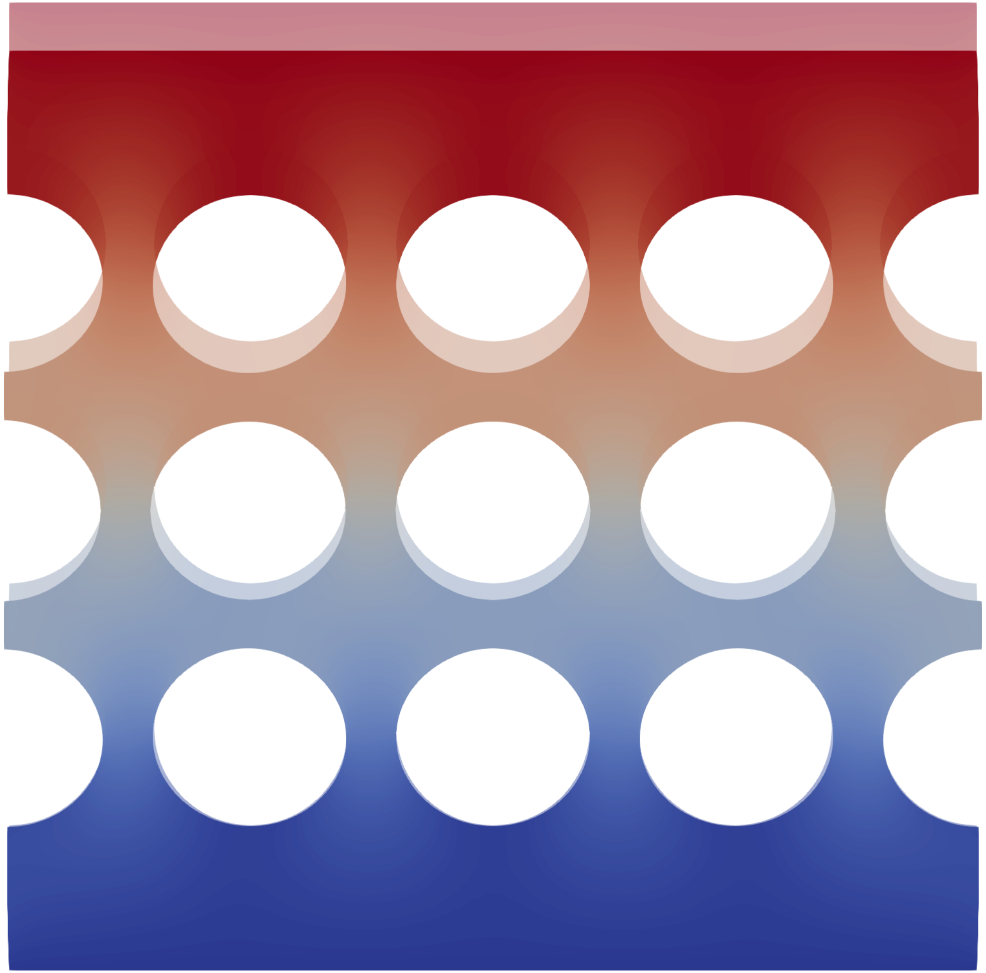
Like linear elasticity, these equations describe how a structure deforms under a load. We set Ω to be the undeformed configuration, and solve for a displacement $u : \Omega \to \mathbb{R}^d$ that describes how each point $X \in \Omega$ maps to the deformed configuration:

$$x(X) = X + u(X).$$

Unlike linear elasticity, hyperelasticity is more realistic because

- (constitutive nonlinearity) the stress-strain curve is not necessarily linear;
- (geometric nonlinearity) the displacements are not necessarily small.

The equations are thus nonlinear.



Challenge!

In this exercise, you will write your own code from scratch.

Good news!

I will tell you everything you need to know.

Section 2

Minimisation ~~and saddle point~~ problems

Many problems can be cast in an optimisation framework.

For example, the Poisson equation arises as the minimisation of the Dirichlet energy

$$J(u) = \frac{1}{2}\int_\Omega \nabla u \cdot \nabla u \;\mathrm{d}x - \int_\Omega f u \;\mathrm{d}x.$$

We can see this by taking its Fréchet derivative and setting it to zero:

$$\begin{aligned}
J_u(u; v) &\coloneqq \lim_{\epsilon \to 0} \frac{J(u + \epsilon v) - J(u)}{\epsilon} \\
&= \lim_{\epsilon \to 0} \frac{1}{\epsilon}\left(\epsilon \int_\Omega \nabla u \cdot \nabla v \;\mathrm{d}x + \epsilon^2 \int_\Omega \nabla v \cdot \nabla v \;\mathrm{d}x - \epsilon \int_\Omega f v \;\mathrm{d}x\right) \\
&= \int_\Omega \nabla u \cdot \nabla v \;\mathrm{d}x - \int_\Omega f v \;\mathrm{d}x = 0,
\end{aligned}$$

the weak statement of the Poisson equation.

We can get Firedrake to do this calculation for us:

```
# Functional to optimise
J = (
      0.5 * inner(grad(u), grad(u))*dx
    -       inner(f, u)*dx
    )

# Calculate the optimality condition (equation to solve)
F = derivative(J, u)
```

Firedrake uses `derivative` inside `solve` to calculate the Jacobian.

Section 3

Hyperelasticity energy functional

Our *dramatis personae*:

- $\Omega \subset \mathbb{R}^d$, the domain;
- $u : \Omega \to \mathbb{R}^d$, the displacement;
- $F = I + \nabla u$, the deformation gradient;
- $C = F^\top F$, the right Cauchy–Green tensor;
- $I_c = \mathrm{tr}(C)$, $J = \det(F)$, invariants;
- μ, λ, Lamé parameters.

note: $J^2 = \det(C) = \mathrm{III}_c$

$\} \Leftarrow$ the strain tensor that matters

Ronald Rivlin

With these, we form the compressible neo-Hookean energy:

$$E(u) = \int_\Omega \frac{\mu}{2}(I_c - d) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2 \,\mathrm{d}x.$$

Stating this in Firedrake:

```
d = mesh.geometric_dimension()
I = Identity(d)
F = I + grad(u)
C = F.T * F
I_c = tr(C)
J = det(F)
```

Section 4

Continuation

Continuation is an extremely powerful algorithm for solving difficult nonlinear problems.

Idea: construct a good initial guess by solving an easier problem.

Continuation

- Solve the problem for easy parameter value.
- While not finished:
  - Use solution for previous parameter as initial guess for next parameter.
  - Increment parameter.

To do continuation in Firedrake, update the parameter in a loop and solve:

```
strain = Constant(0)  # placeholder Constant
# Use the strain as our boundary condition value:
bcs = [... ,
       DirichletBC(V.sub(1), strain, top),
       ...]

strains = ...
for strain_ in strains:
    strain.assign(strain_)  # update parameter value
    solve(F == 0, u, bcs)   # solve for next parameter
```

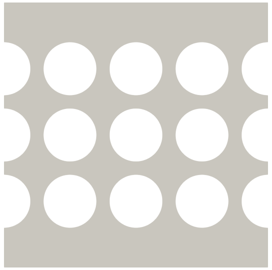
Challenge!

Solve the equations of hyperelasticity on the domain

$$\Omega = (0,1)^2 \setminus \left(\bigcup_{ij} D_{ij} \right),$$

where $i \in \{1, \ldots, 3\}, j \in \{1, \ldots, 5\}$, and

$$D_{ij} = \left\{ (x, y) \in \mathbb{R}^2 : \left(x - \frac{j-1}{4} \right)^2 + \left(y - \frac{i}{4} \right)^2 \leq 0.1^2 \right\}.$$



Challenge!

Solve the problem with $\mu = 4 \times 10^5$, $\lambda = 6 \times 10^5$, and boundary conditions

$$\begin{aligned} u &= (0, 0) \quad \text{on } \{y = 0\}, \\ u &= (0, -s) \text{ on } \{y = 1\}, \end{aligned}$$

for $s = 0.1$.

Apply natural (i.e. do-nothing, stress-free) boundary conditions on all other boundaries.

Hint: you will probably need to employ continuation.

```python
from firedrake import *
from netgen.occ import *
from numpy import linspace

# Build 3x3 mesh of holes
rect = WorkPlane(Axes((0,0,0), n=Z, h=X)).Rectangle(1,1).Face().bc("sides")
rect.edges.Min(Y).name = "bottom"
rect.edges.Max(Y).name = "top"

shape = rect
for i in range(1, 4):
    for j in range(1, 6):
        centre_x = 0.25*(j-1)
        centre_y = 0.25*(i+0)
        disk = WorkPlane(Axes((centre_x, centre_y, 0), n=Z, h=X)).Circle(0.1).Face()
        shape = shape - disk

geo = OCCGeometry(shape, dim=2)
ngmesh = geo.GenerateMesh(maxh=1)
base = Mesh(ngmesh)
mh = MeshHierarchy(base, 2, netgen_flags={})
mesh = mh[-1]
d = mesh.geometric_dimension()

bottom = [i + 1 for (i, name) in
          enumerate(ngmesh.GetRegionNames(codim=1)) if name == "bottom"]
top    = [i + 1 for (i, name) in
          enumerate(ngmesh.GetRegionNames(codim=1)) if name == "top"]

V = VectorFunctionSpace(mesh, "CG", 2)
u = Function(V, name="Displacement")

# Kinematics
I = Identity(d)  # Identity tensor
F = I + grad(u)  # Deformation gradient
C = F.T*F        # Right Cauchy-Green tensor

# Invariants of deformation tensors
Ic = tr(C)
J  = det(F)

# Elasticity parameters
mu = Constant(400000)
lmbda = Constant(600000)
print(f"μ: {float(mu)}")
print(f"λ: {float(lmbda)}")

# Stored strain energy density (compressible neo-Hookean model)
psi = (mu/2)*(Ic - d) - mu*ln(J) + (lmbda/2)*(ln(J))**2

# Total potential energy
J = psi*dx

# Hyperelasticity equations. Quite hard to write down!
R = derivative(J, u)

# Boundary conditions
strain = Constant(0)
bcs = [DirichletBC(V, Constant((0, 0)), bottom),
       DirichletBC(V.sub(0), 0, top),
       DirichletBC(V.sub(1), strain, top)]

sp = {"snes_monitor": None, "snes_linesearch_type": "l2"}
#sp = {"snes_monitor": None}

pvd = File("output/hyperelasticity.pvd")
pvd.write(u, time=0)
for strain_ in linspace(0, -0.1, 41)[1:]:
    print(f"Solving for strain {strain_:.4f}")
    strain.assign(strain_)
    solve(R == 0, u, bcs, solver_parameters=sp)
    pvd.write(u, time=-strain_)
```

repo:

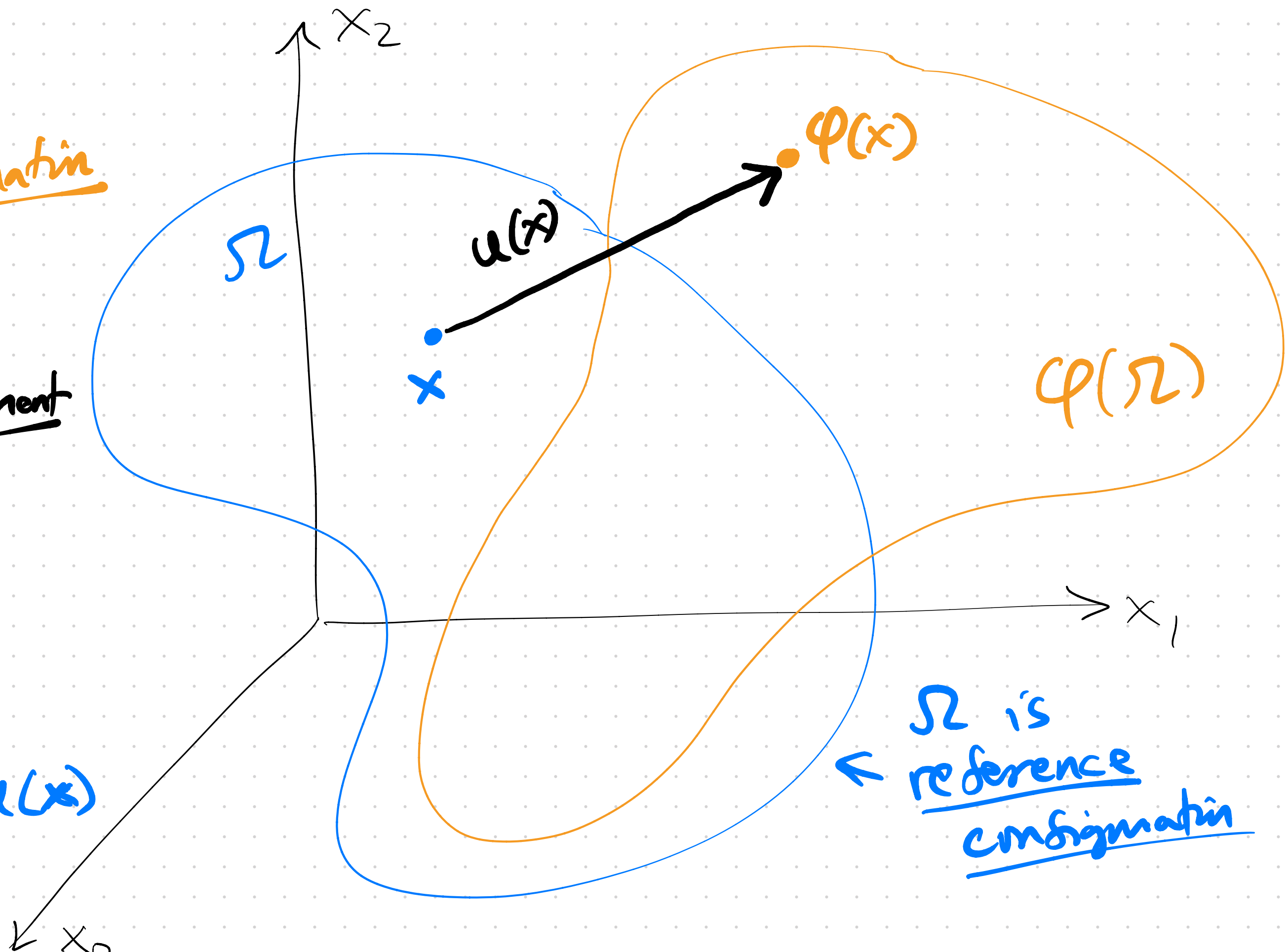
github.com/pefarrell/icerm2024

- there is a lot to unpack here!
- <u>agenda:</u>
  1. Poisson equation code using only energy objective functional } code poisson.py
  2. neo-Hookean hyperelasticity
     - in theory } start with 2 slides →
     - in the code
  3. continuation
  4. Netgen and Open Cascade meshing
  5. running the code and visualizing output

code hyperelasticity.py

$\varphi : \Omega \to \mathbb{R}^3$ is <u>deformation</u>

$u : \Omega \to \mathbb{R}^3$ is <u>displacement</u>



Ω is ← <u>reference configuration</u>

<u>note:</u>

$\varphi(x) = x + u(x)$

$\nabla\varphi = I + \nabla u$

↑ called "F" in elasticity literature

def:

$$C(x) = \nabla\varphi(x)^T \nabla\varphi(x) = F^T F$$

is the right Cauchy-Green strain tensor

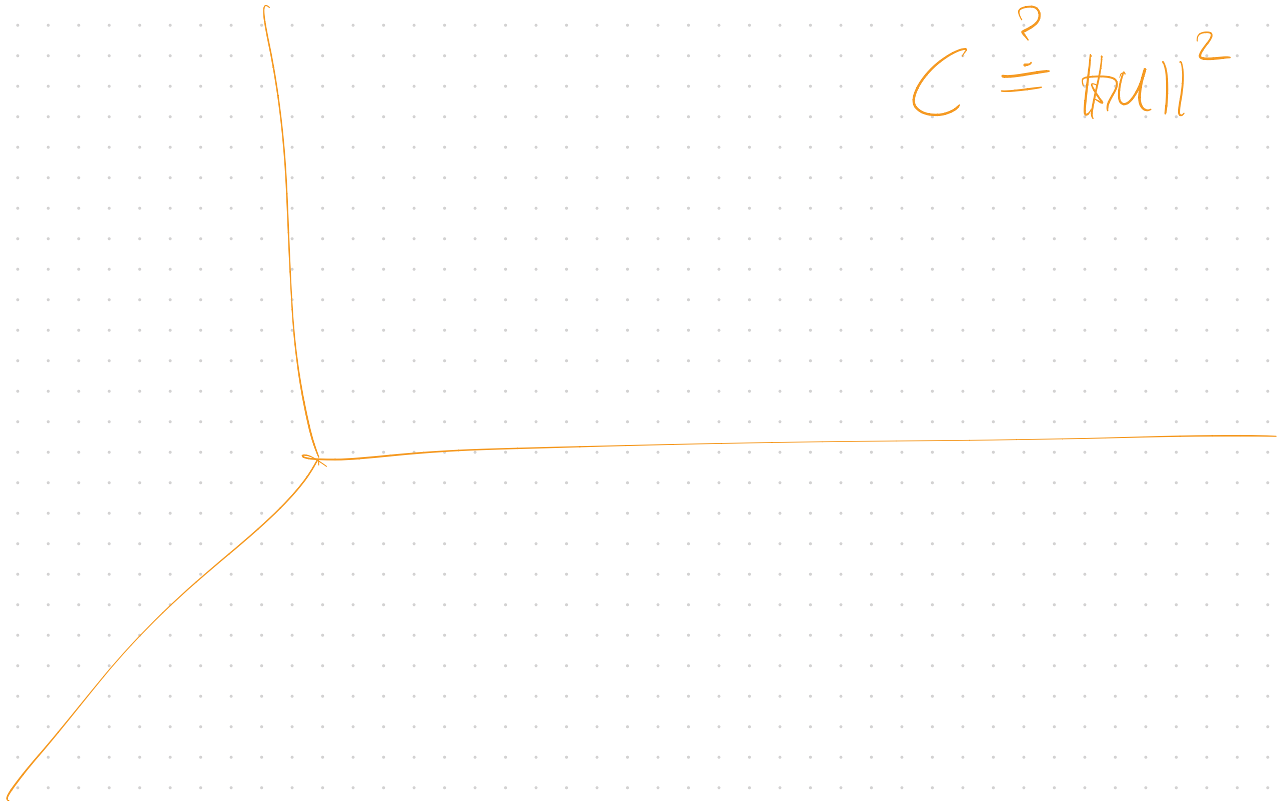
$\to$ $$C(x) = (I + \nabla u)^T (I + \nabla u) = I + \nabla u + \nabla u^T + \nabla u^T \nabla u$$

def:

$$E(x) = \frac{1}{2}(C(x) - I)$$

is the strain tensor field, a.k.a. the Green-St. Venant strain tensor

$\to$ $$E(u) = \frac{1}{2}\left(\nabla u(x) + \nabla u(x)^T + \nabla u(x)^T \nabla u(x)\right)$$



$$C \stackrel{?}{=} \|\pi u\|^2$$

def:

$$\varepsilon(u) = \frac{1}{2}\left(\nabla u(x) + \nabla u(x)^T\right)$$

is the linearized strain tensor

not used in this example