Developing simulation technology to solve biomedical problems: analysis, implementation and applications

Marie E. Rognes
Center for Biomedical Computing
Simula Research Laboratory

Eleonora Piersanti, Andre Massing, FEniCS and Dolfin-adjoint teams

Outline

1. Introduction to CBC and BioComp at Simula
2. Robust numerical methods
   - Fictitious domain method for Stokes equations
   - Mixed finite elements for multiple-network poroelasticity
3. Automated software for solving partial differential equations
   - Introduction to the FEniCS Project
   - Automated goal-oriented error control
   - Dolfin-adjoint: automated derivation of discrete adjoints

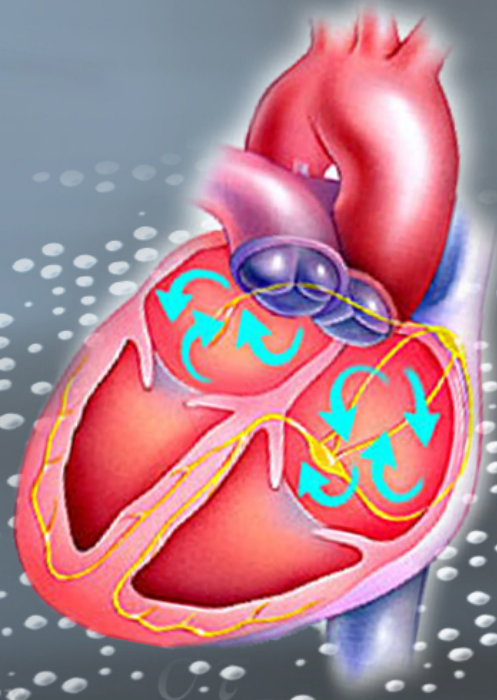
Introduction to CBC and BioComp

Norwegian Center of Excellence

Center for Biomedical Computing

2007 - 2017

```
M = G.compute_mass_matrix(velocity_element)
A = G.compute_stiffness_matrix(velocity_element)
B = G.compute_div_matrix(velocity_element, pressure_element)
D = G.compute_stiffness_matrix(pressure_element)

T = 1; dt = 0.1
while t < T:
  t = t + dt;
  f = G.compute_source_vector(rhs)
  C = G.compute_convection_matrix(velocity_element, v_prev)

  A1 = M + dt*A + dt*C
  prec1 = MLPrec(A1)
  v, iter = precondBiCGStab(prec1, A1, v, f , 1.0e-9)

  g = (1.0/dt)*B.t*v
  phi, iter = precondConjGrad(MLPrec(D), D, phi, g , 1.0e-9)
  v = v - dt*B.t*phi
  p = (Mp + dt*Ap)phi
```



$$\rho\left(\frac{\partial u}{\partial t} + V \cdot \nabla u\right) = -\frac{\partial}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x$$

CENTER FOR CARDIOLOGICAL **INNOVATION**

sfi = Centre for Research-based Innovation

Host institution:

Dept of Card.
Oslo University Hospital, Rikshospitalet

Partners:

Scientific:

- Simula Research Laboratory
- UoO

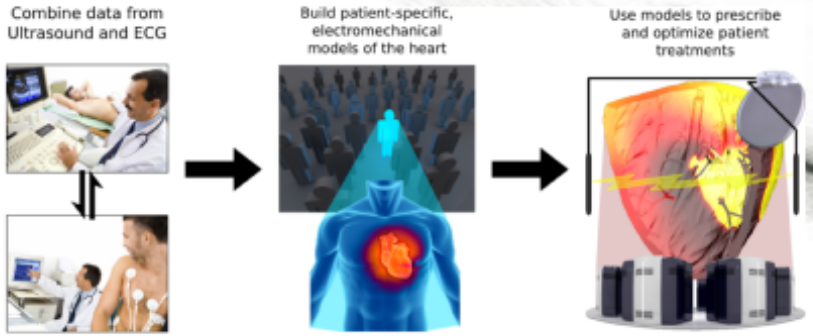
Industrial:

- GE Vingmed Ultrasound AS
- CardioSolv LLC
- Kalkulo AS
- Medtronic

Purpose:

- next generation of ultrasound systems
- advanced patient-specific computer simulation
- multi-modality visualization techniques
- wireless ECG in ultrasound
- optimizing lead placement CRT

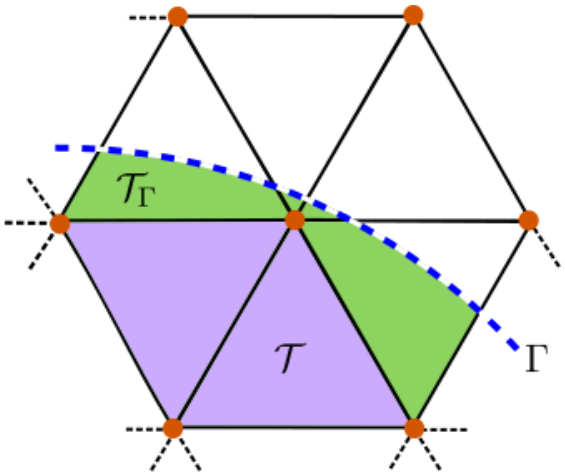
Sudden Cardiac Death
Heart failure



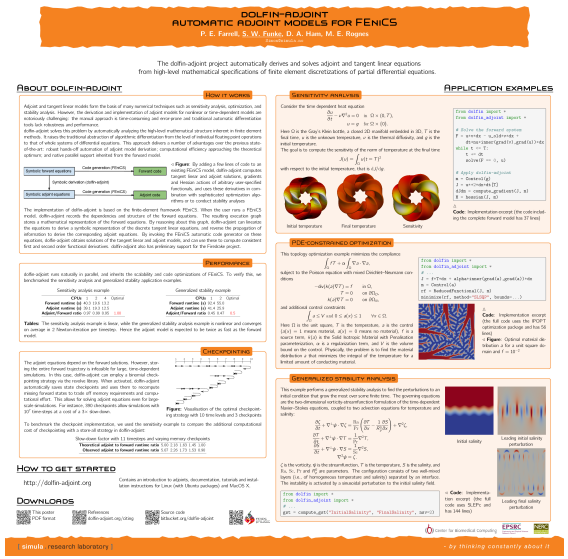
Oslo universitetssykehus · GE Healthcare · Medtronic · [simula . research laboratory] · kalkulo · cardiosolv

Developing the next generation simulation technology to solve problems affecting human health and disease

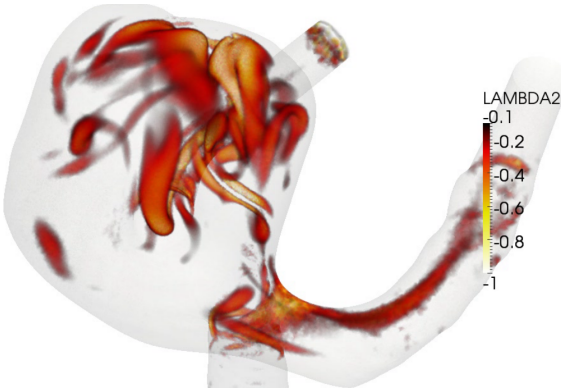
The Biomedical Computing department @ Simula



[Massing et al., Numer. Math., 2014]



[Farrell et al., SISC, 2013]



[Valen-Sendstad et al, 2015]

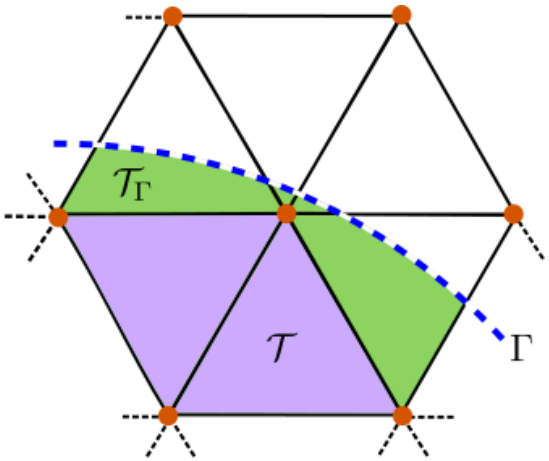
Center for Biomedical Computing

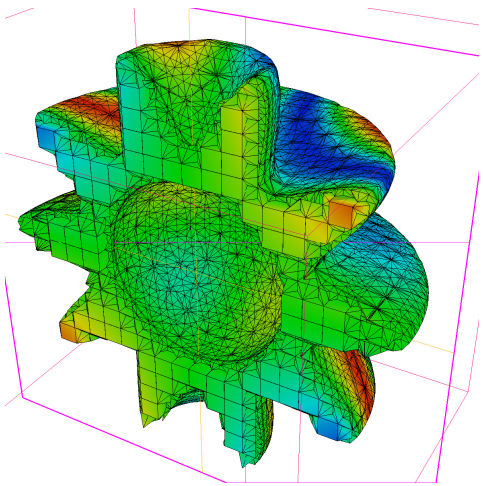
Norwegian Centre of Excellence

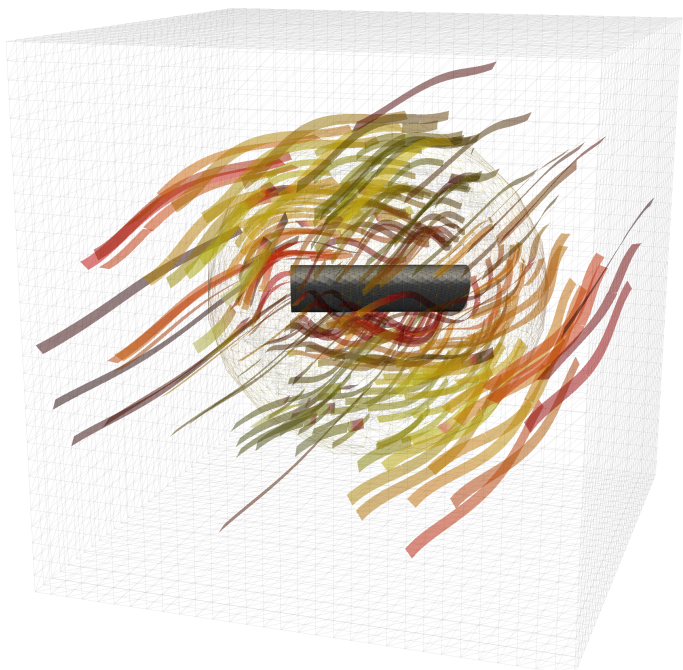
Robust numerical methods

Cut finite element methods for viscous flow
(with A. Massing, A. Logg, M. G. Larson)

Multi-mesh finite element methods offers geometrical flexibility and robustness





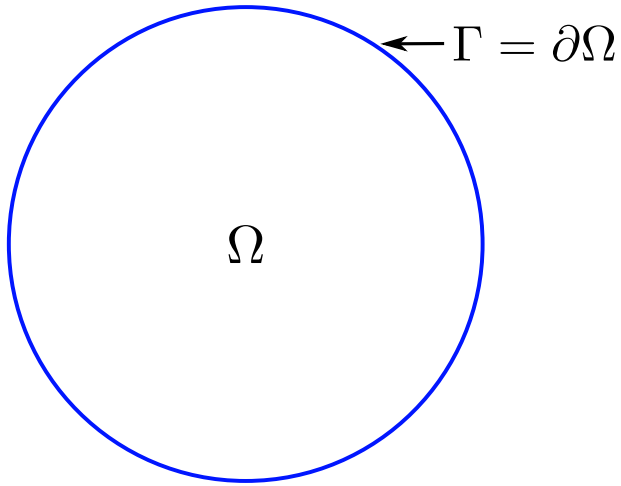


[Johansson and Larson, 2013; Massing et al, 2013, 2014, 2015]

Classical stabilized finite element formulation of Stokes

Find the velocity $u : \Omega \subset \mathbb{R}^d \to \mathbb{R}^d$ and the pressure $p : \Omega \to \mathbb{R}$ s.t:

$$\begin{aligned} -\Delta u + \nabla p &= f \quad \text{in } \Omega, \\ \nabla \cdot u &= 0 \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma, \end{aligned}$$



Stabilized FEM over conforming mesh

Find $u_h \in V_h$ and $p_h \in Q_h$ such that

$$a(u_h, v) + b(u_h, q) + b(v, p_h) - c_h(p_h, q) = L_h(v, q)$$

for all $v \in V_h$ and $q \in Q_h$ where $a(u, v) = \langle \operatorname{grad} u, \operatorname{grad} v \rangle_\Omega$ and $b(u, v) = -\langle \operatorname{div} u, q \rangle_\Omega$ and

$$c_h(p, q) = \begin{cases} \beta_0 \displaystyle\sum_{F \in \partial_i \mathcal{T}} h_F \langle [\![p]\!], [\![q]\!] \rangle_F & \text{if } Q_h = P^0(\mathcal{T}) \\ \beta_1 \displaystyle\sum_{T \in \mathcal{T}} h_T^2 \langle \operatorname{grad} p, \operatorname{grad} q \rangle_T & \text{if } Q_h = P^1(\mathcal{T}) \end{cases}$$

Stabilized Nitsche finite element formulation of Stokes

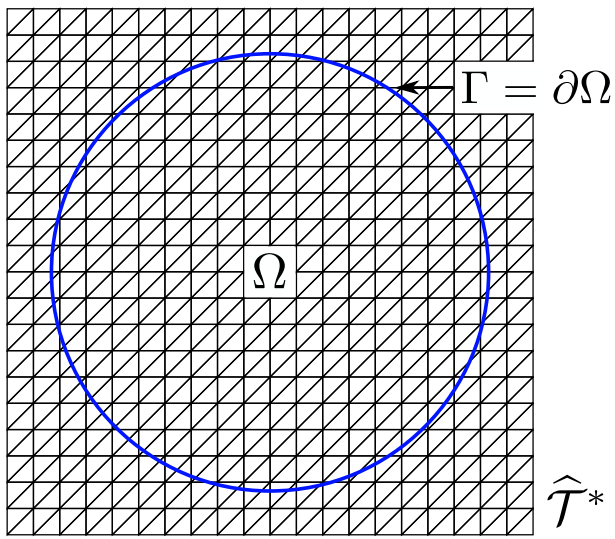
Find $u_h \in V_h$ and $p_h \in Q_h$ such that

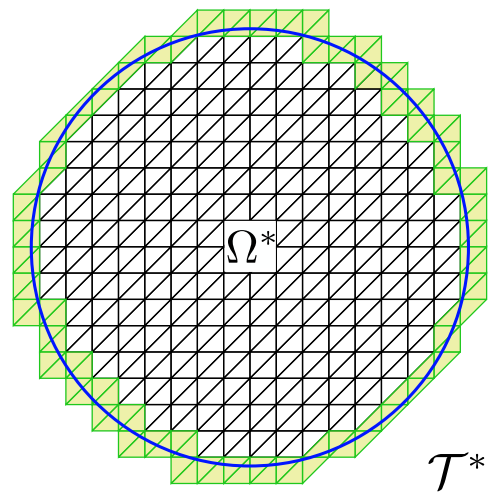
$$a_h(u_h, v) + b_h(u_h, q) + b_h(v, p_h) - c_h(p_h, q) + J = G_h(v, q)$$

for all $v \in V_h$ and $q \in Q_h$ where

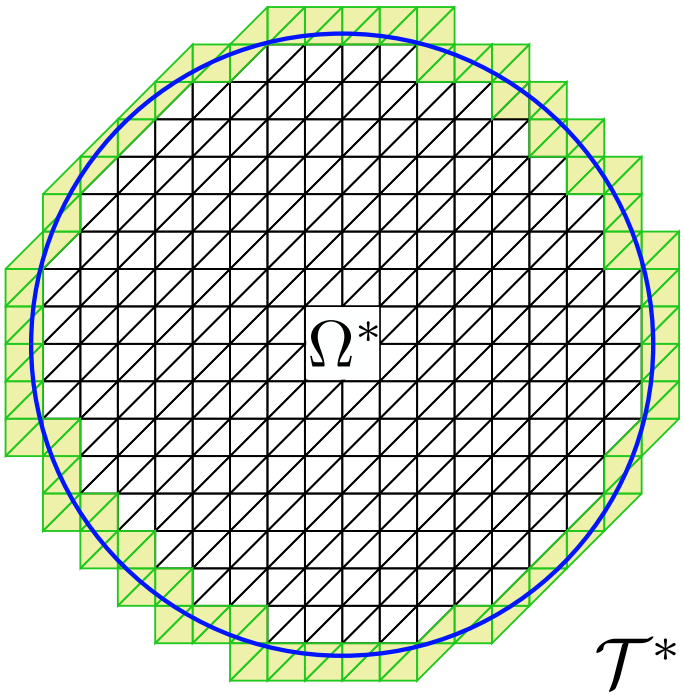
$$a_h(u, v) = \langle \operatorname{grad} u, \operatorname{grad} v \rangle_\Omega - \langle \partial_n u, v \rangle_\Gamma - \langle \partial_n v, u \rangle_\Gamma + \gamma \langle h^{-1} u, v \rangle_\Gamma$$
$$b_h(u, q) = -\langle \operatorname{div} u, q \rangle_\Omega + \langle n \cdot u, q \rangle_\Gamma$$

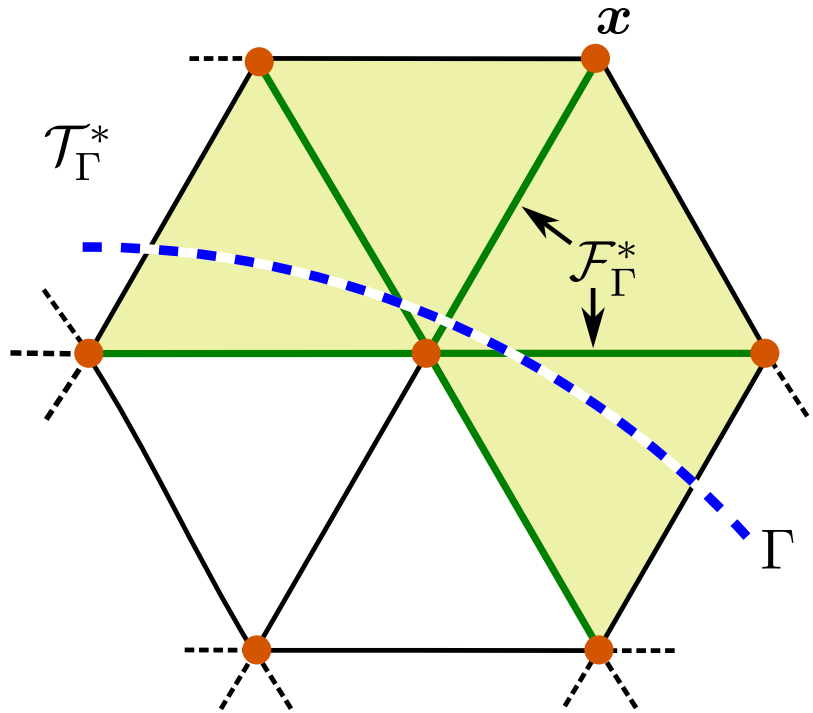
and $J = J(u_h, p_h, v, q) = i(u_h, v) - j(p_h, q)$ are fictitious domain terms.





The fictitious domain method can be made robust by adding penalties acting in the boundary zone

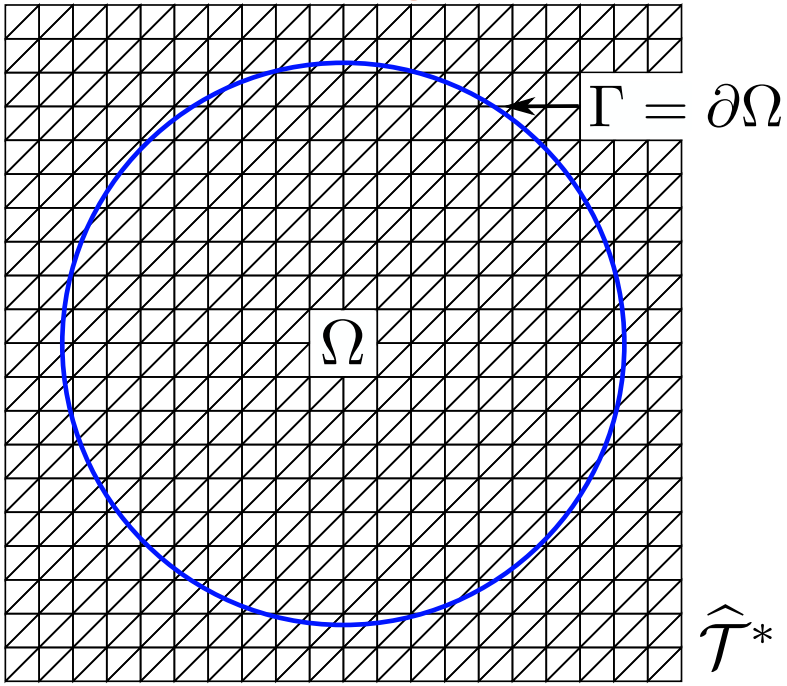


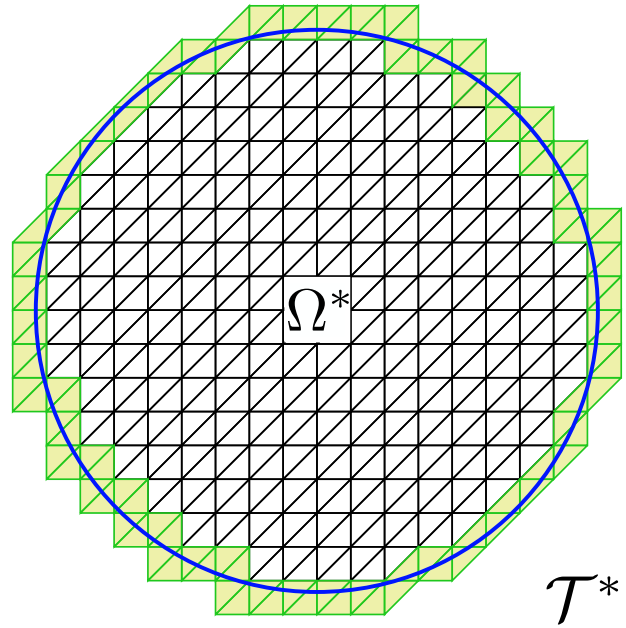


$$i_h(u_h, v_h) = \beta_2 \sum_{F \in \mathcal{F}_\Gamma^*} h_F([\partial_n u_h], [\partial_n v_h])_F,$$

$$j_h(p_h, q_h) = \begin{cases} \beta_0 \displaystyle\sum_{F \in \mathcal{F}_\Gamma^*} h_F([\![p_h]\!], [\![q_h]\!])_F & \text{if } Q_h = P^0(\mathcal{T}^*), \\ \beta_3 \displaystyle\sum_{F \in \mathcal{F}_\Gamma^*} h_F^3([\![\partial_n p_h]\!], [\![\partial_n q_h]\!])_F & \text{if } Q_h = P^1(\mathcal{T}^*). \end{cases}$$

Boundary zone penalties allows for reconstruction of norms on the entire background mesh $\mathcal{T}^*$





$$\|\nabla v_h\|^2_{\Omega^*} \lesssim \|\nabla v_h\|^2_{\Omega} + \sum_{F\in\mathcal{F}^*_\Gamma} h_F([\![\partial_n v_h]\!],[\![\partial_n v_h]\!])_F \lesssim \|\nabla v_h\|^2_{\Omega^*} \quad \forall\, v_h \in V_h(\mathcal{T}^*),$$

$$\|q_h\|^2_{\Omega^*} \lesssim \|q_h\|^2_{\Omega} + \sum_{F\in\mathcal{F}^*_\Gamma} h_F([\![n\cdot q_h]\!],[\![n\cdot q_h]\!])_F \lesssim \|q_h\|^2_{\Omega^*} \quad \forall\, q_h \in P^0(\mathcal{T}^*),$$

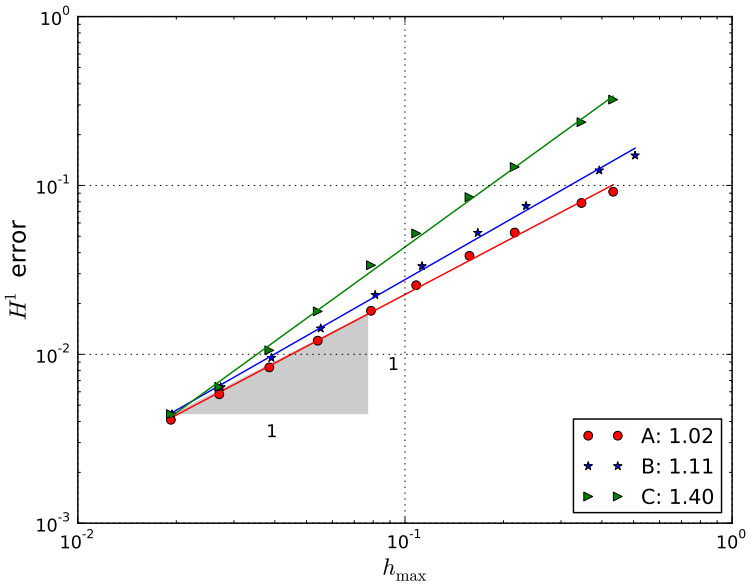
$$\|q_h\|^2_{\Omega^*} \lesssim \|q_h\|^2_{\Omega} + \sum_{F\in\mathcal{F}^*_\Gamma} h^3_F([\![\partial_n q_h]\!],[\![\partial_n q_h]\!])_F \lesssim \|q_h\|^2_{\Omega^*} \quad \forall\, q_h \in P^1(\mathcal{T}^*).$$

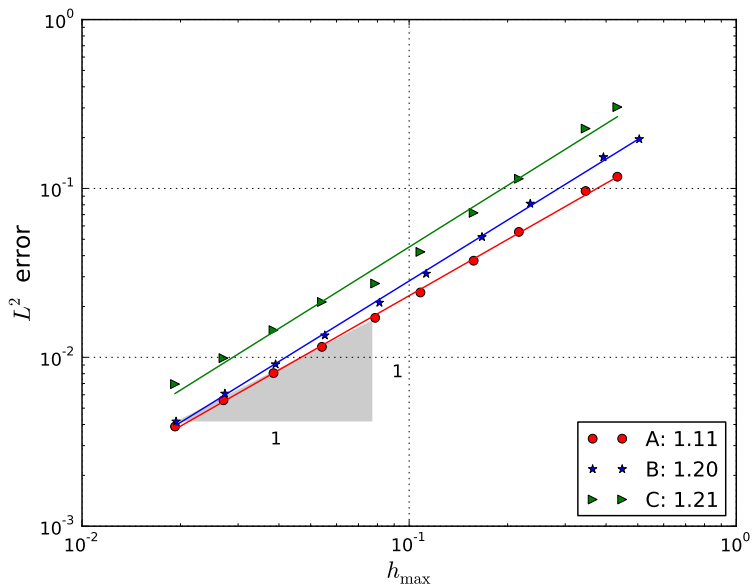
The proposed fictitious domain method satisfies an optimal *a priori* estimate

Theorem

Let $(u, p) \in [H^2(\Omega)]^d \times H^1(\Omega)$ *be the solution of the Stokes problem and let* (u_h, p_h) *be the discrete solution of the stabilized Nitsche fictitious domain formulation. Then*

$$|||(u - u_h, p - p_h)||| \lesssim h \left(|u|_{2,\Omega} + |p|_{1,\Omega} \right).$$



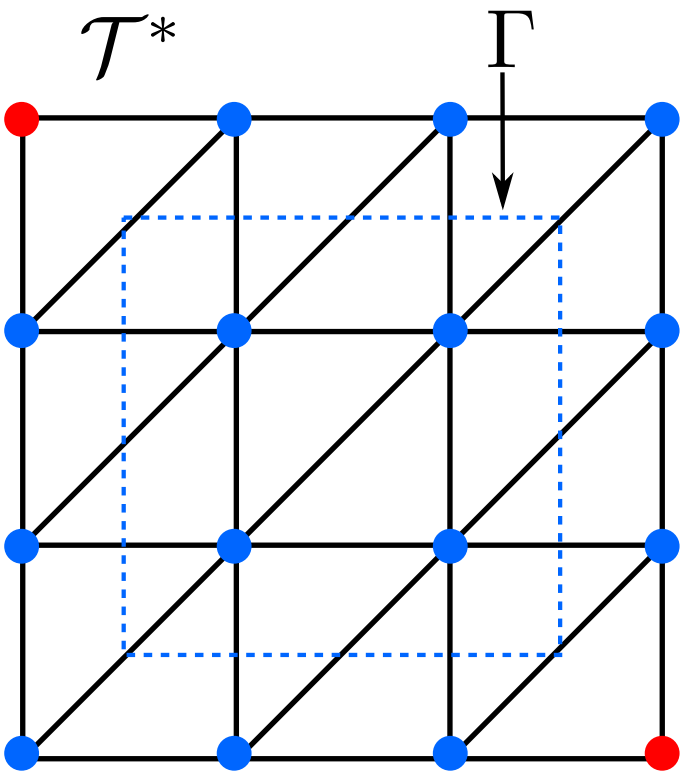


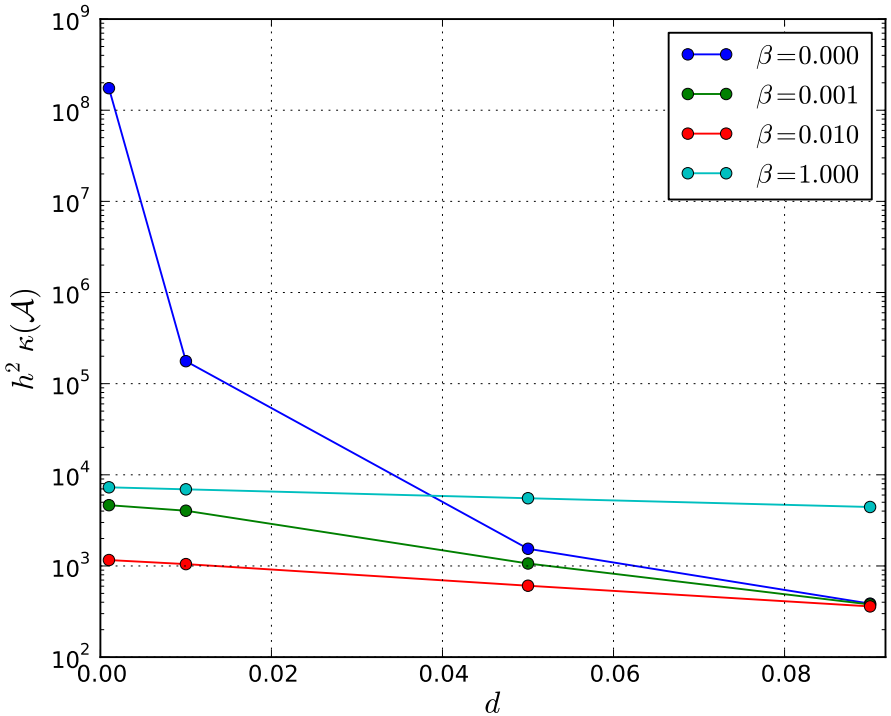
[Massing, Logg, Larson, R. (2012)]

The condition number can be bounded independently of the location of the boundary

There is a C independent of the position of Γ, s.t. the condition number of the stiffness matrix $\mathcal{A}$ associated with the Nitsche fictitious domain method satisfies

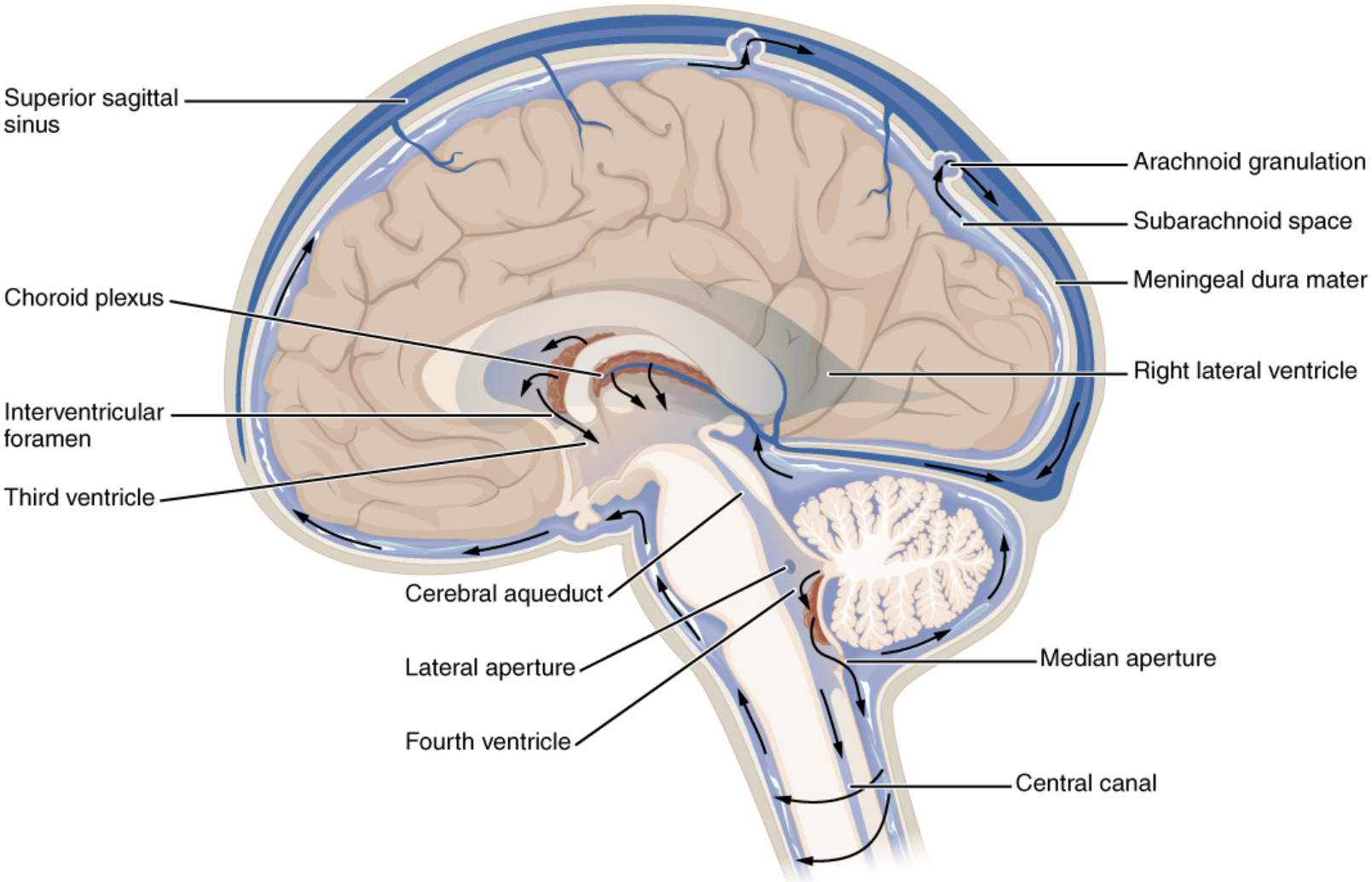
$$\kappa(\mathcal{A}) \leqslant Ch^{-2},$$





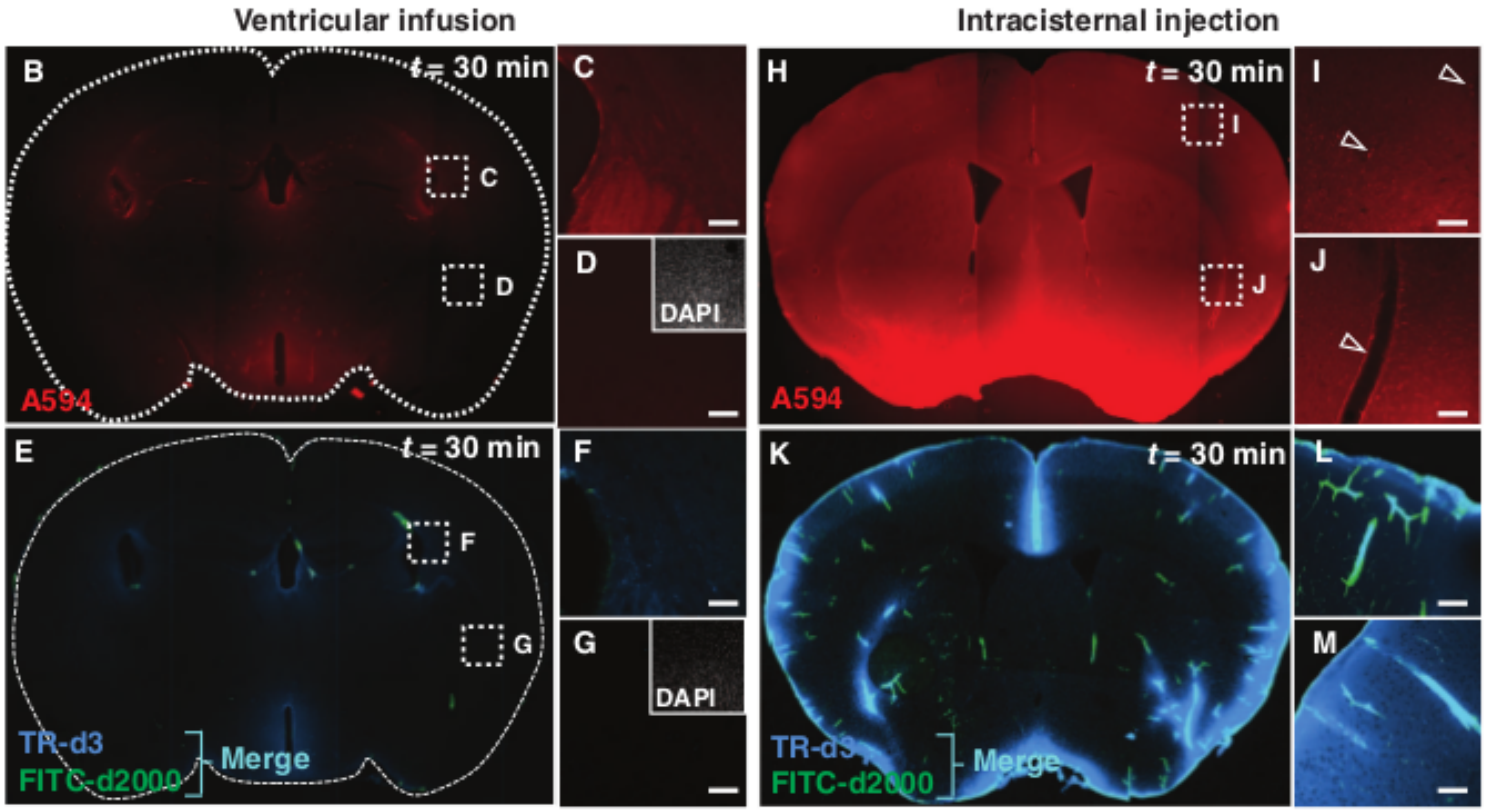
Mixed finite element methods for porous media
(with J. Lee, E. Piersanti, K.-A. Mardal)

The cerebrospinal fluid (CSF) circulates in the subarachnoid space around the brain and possibly within



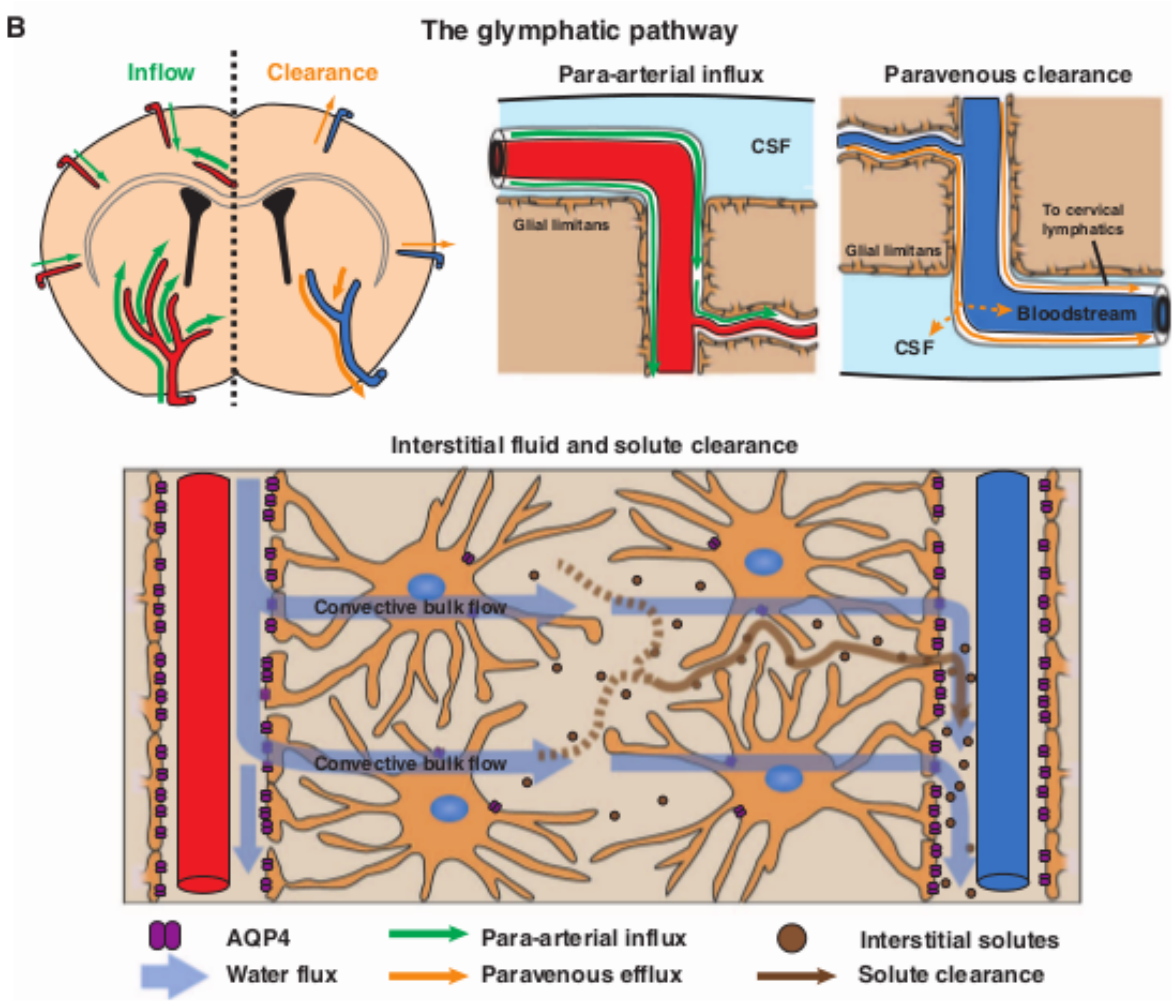
[Image source: Wikimedia]

Paravascular pathways facilitate CSF bulk flow through the brain parenchyma



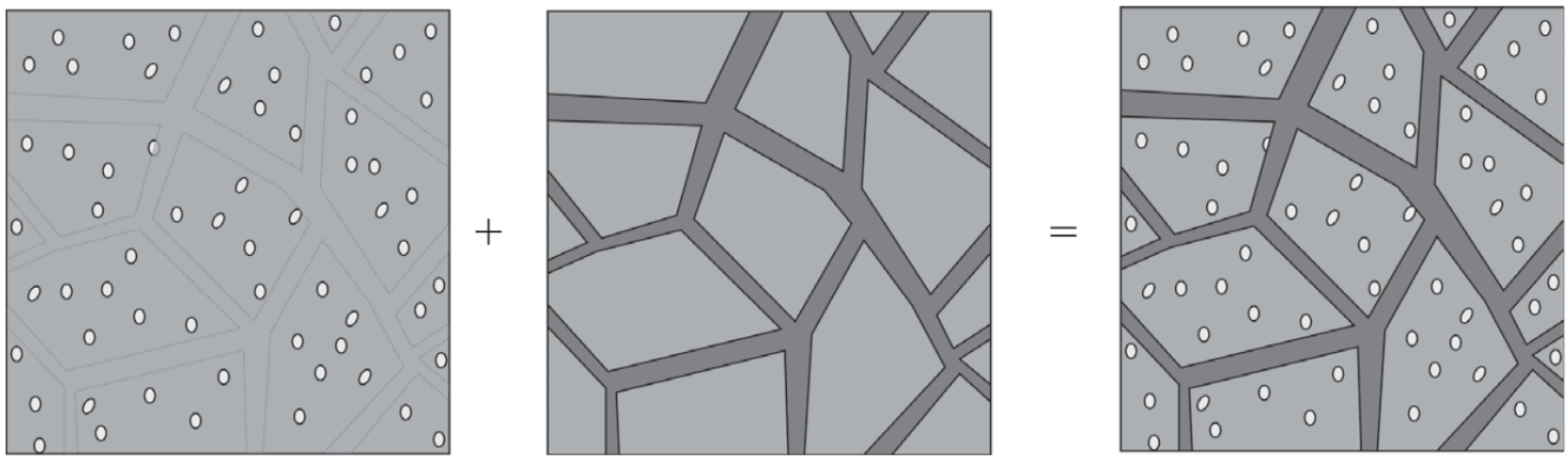
[Iliff et al., A Paravascular Pathway Facilitates CSF Flow Through the Brain Parenchyma and the Clearance of Interstitial Solutes, Including Amyloid-β, Sci Trans Med, 2012]

Paravascular pathways facilitate CSF bulk flow through the brain parenchyma



[Iliff et al., A Paravascular Pathway Facilitates CSF Flow Through the Brain Parenchyma and the Clearance of Interstitial Solutes, Including Amyloid-β, Sci Trans Med, 2012]

Multiple-network poroelastic theory (MPET) is a macroscopic model for poroelastic media with multiple fluid networks



[Tully and Ventikos, Jour. of Fluid Mech. 2011]

The multiple-network poroelasticity theory extends on Biot's equations

Find $u = u(x,t)$ and the pressures $p_a = p_a(x,t)$ for $a = 1, \ldots, A$ such that

$$-\operatorname{div}(\sigma^*) + \sum_a \alpha_a \operatorname{grad} p_a = f, \tag{2}$$

$$c_a \dot{p}_a + \alpha_a \operatorname{div} \dot{u} - \operatorname{div} K_a \operatorname{grad} p_a + S_a = g_a, \quad a = 1, \ldots, A \tag{3}$$

Fluid is transferred between networks, e.g.:

$$S_a = \sum_b s_{b\to a}, \quad s_{b\to a} = \xi_{ab}(p_b - p_a),$$

Elastic tissue deforms linearly but potentially anisotropically, e.g.:

$$\sigma^*(u) = C\varepsilon(u),$$

[Bai, Elsworth and Roegiers, Water Resources Research, 1993; Tully and Ventikos, Jour Fluid Mech., 2011]

Standard finite element formulation is not robust

Find $u \in V_h \subseteq [H_0^1(\Omega)]^d$ and $p_a \in Q_h \subseteq [H_0^1(\Omega)^d$ for $a = 1, \ldots, A$ such that

$$\langle \sigma^*, \operatorname{grad} v \rangle + \sum_a \langle \alpha_a p_a, \operatorname{div} v \rangle = \langle f, v \rangle$$

$$\langle c_a \dot{p}_a + \alpha_a \operatorname{div} \dot{u} + S_a, q_a \rangle + \langle K_a \operatorname{grad} p_a, \operatorname{grad} q_a \rangle = \langle g_a, q_a \rangle$$

for all $v \in V_h$, $q \in Q_h$.

Relevant parameter regimes

- $500 \leq \lambda \leq 10^6$ (Pa)
- $K \in (10^{-10}, 10^{-7})$ $(\frac{\mathrm{m}^4}{(Ns)})$
- $S \to 0$, $c_a \to 0$.
- $h, \Delta t \to 0$

[Ben-Hatira et al, J. Biomed. Sci. and Engrg. (2012); Vardakis et al, Med. Engrg. & Phys. (2016)]

Symptoms

- Loss of convergence as $\lambda \to \infty$
- Oscillations for large variations in K
- Condition number growth $(K, \lambda, \Delta t)$

By introducing a total pressure, we obtain a more attractive formulation

Find $u \in V_h \subseteq [H_0^1(\Omega)]^d$ and $p_a \in Q_h \subseteq [H_0^1(\Omega)^d$ for $a = 1, \ldots, A$ and $p \in Q_h$ such that ($\alpha_a = 1$ for brevity)

$$\langle p - \lambda \operatorname{div} u + \sum_a p_a, q \rangle = 0$$

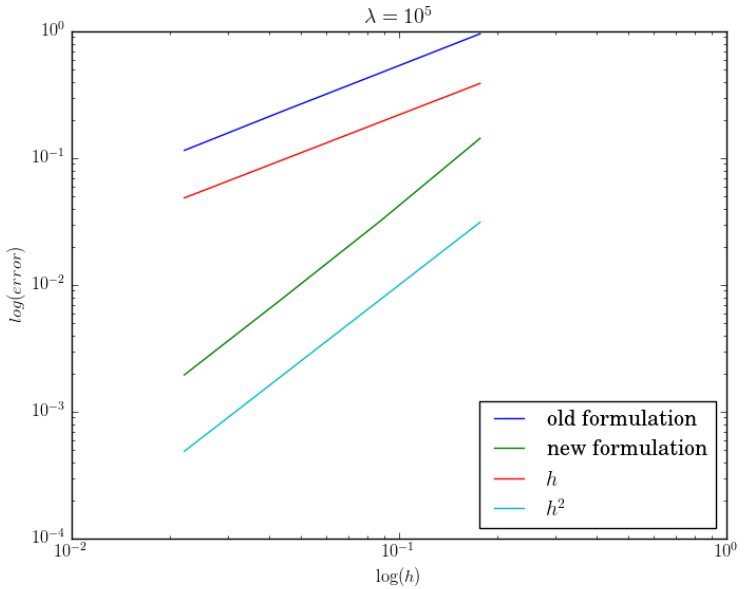
$$\langle 2\mu\epsilon(u), \operatorname{grad} v \rangle + \langle p, \operatorname{div} v \rangle = \langle f, v \rangle$$

$$\langle c_a \dot{p}_a - \frac{1}{\lambda} \left(\dot{p} - \sum_a \dot{p}_a \right) + S_a, q_a \rangle + \langle K_a \operatorname{grad} p_a, \operatorname{grad} q_a \rangle = \langle g, q_a \rangle$$

for all $v \in V_h$, $q_a \in Q_h$, $q \in Q_h$.

[Lee, Piersanti, Mardal, R., in preparation, 2016]

The total pressure formulation restores convergence and allows for robust block preconditioning



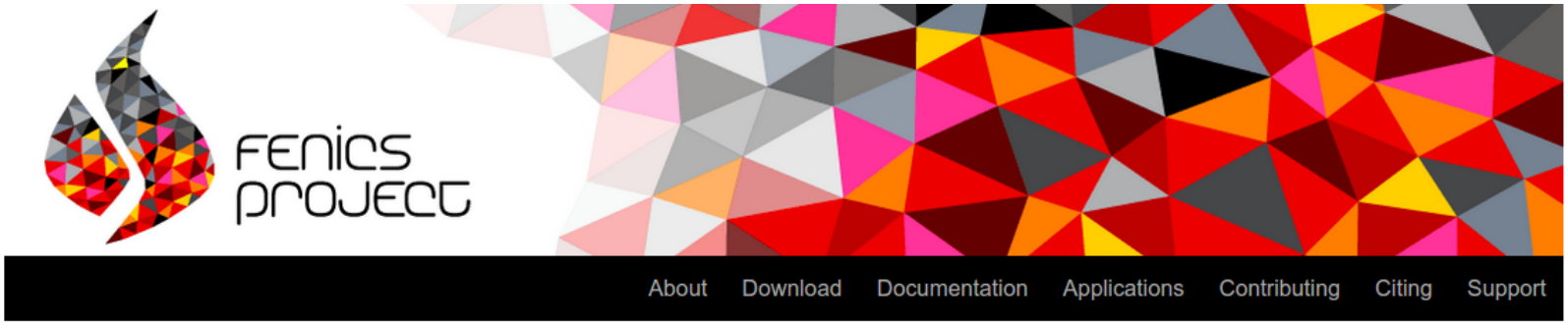
Smooth test case with $A = 2$, Crank-Nicolson in time, parameter-dependent block preconditioner, PETSc fieldsplit solver.

		N			
λ	$K_{1,2}$	8	16	32	64
10^0	10^0	56	57	56	55
	10^{-3}	60	61	65	68
	10^{-5}	60	61	65	65
	10^{-8}	61	61	64	65
10^3	10^0	68	72	73	73
	10^{-3}	70	72	73	73
	10^{-5}	71	73	72	73
	10^{-8}	71	72	72	53
10^5	10^0	78	52	53	54
	10^{-3}	52	54	50	54
	10^{-5}	53	54	54	55
	10^{-8}	54	52	52	55

[Lee, Piersanti, Mardal, R., in preparation, 2016]

Automated software for solving partial differential equations

The FEniCS Project



The FEniCS Project is a collection of free software with an extensive list of features for automated, efficient solution of differential equations.

Through this web site, you can learn more about the project and learn how to obtain and how to use our software. We'd be delighted to offer support in case you need it, and encourage contributions from our users.

[fenicsproject.org]

- FEniCS is an international open source software and research project
- FEniCS is user-friendly: estimated $10^3 - 10^4$ users world-wide
- FEniCS is efficient: parallel performant up to (at least) $25\,000$ cores.

FEniCS code can be readable, scale with mathematical complexity, and provide high-performance

Stokes with nonlinear viscosity

Given temperature T, find velocity u and pressure p such that

$$-\operatorname{div}(2\nu(u,T)\varepsilon(u) + pI) = \operatorname{Ra} T\, g$$
$$\operatorname{div} u = 0$$

in Ω with (for instance)

$$\nu(u,T) = e^{-\alpha T}\,(u \cdot u).$$

Finite element formulation

Given temperature T, find $(u,p) \in W = V \times Q$ such that

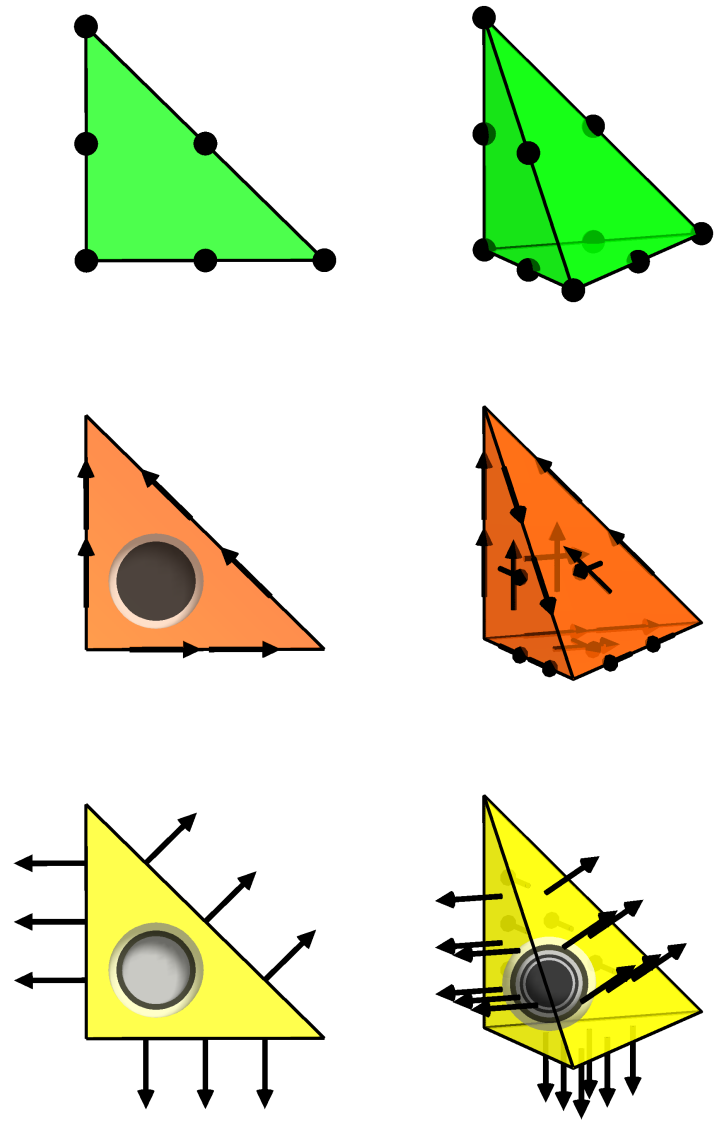
$$\int_\Omega 2\nu(u,T)\varepsilon(u) \cdot \varepsilon(v) + \operatorname{div}(v)\, p + \operatorname{div}(u)\, q - \operatorname{Ra} T g \cdot v \,\mathrm{d}x = 0$$

for all $(v,q) \in W$.

```
from dolfin import *

# Define viscosity
def nu(u, T):
  return exp(-10.0*T)*dot(u, u)

# Define element spaces
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V * Q

# Define functions
T = Expression("...")
w = Function(W)
(u, p) = split(w)
(v, q) = TestFunctions(W)

# Define equation
F = (2*nu(u, T)*inner(eps(u), eps(v))
     + div(v)*p + div(u)*q
     + Ra*T*v[1])*dx
bcs = ...

# Solve F = 0 w.r.t w
solve(F == 0, w, bcs)
```

FEniCS provides a wide range of (mixed) finite element spaces

```
from dolfin import *

# Define viscosity
def nu(u, T):
  return exp(-10.0*T)*dot(u, u)

# Define element spaces
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V * Q

# Define functions
T = Expression("...")
w = Function(W)
(u, p) = split(w)
(v, q) = TestFunctions(W)

# Define equation
F = (2*nu(u, T)*inner(eps(u), eps(v))
     + div(v)*p + div(u)*q
     + Ra*T*v[1])*dx
bcs = ...

# Solve F = 0 w.r.t w
solve(F == 0, w, bcs)
```

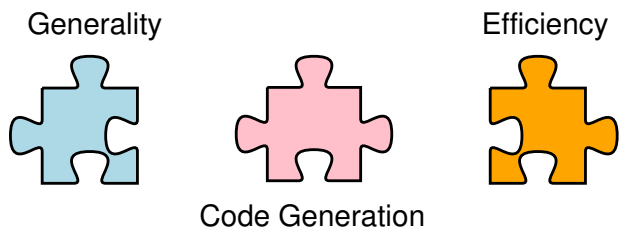


Kirby 2004; Ølgaard, Logg, and Wells 2008; R., Kirby and Logg 2009

FEniCS provides an expressive form language close to mathematical syntax

```
from dolfin import *

# Define viscosity
def nu(u, T):
  return exp(-10.0*T)*dot(u, u)

# Define element spaces
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V * Q

# Define functions
T = Expression("...")
w = Function(W)
(u, p) = split(w)
(v, q) = TestFunctions(W)

# Define equation
F = (2*nu(u, T)*inner(eps(u), eps(v))
     + div(v)*p + div(u)*q
     + Ra*T*v[1])*dx
bcs = ...

# Solve F = 0 w.r.t w
solve(F == 0, w, bcs)
```

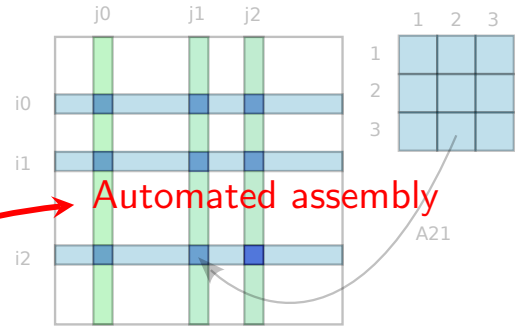
Language for variational forms



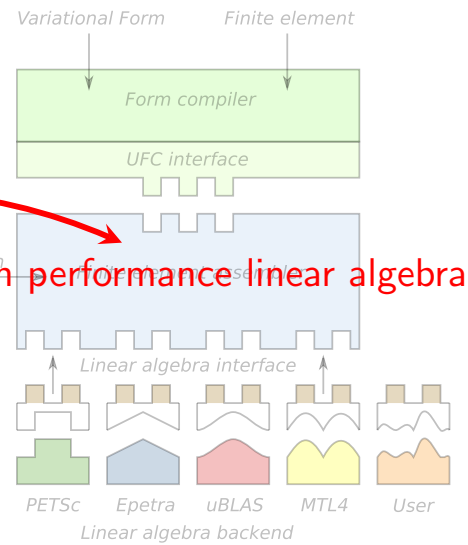
Kirby and Logg 2006; Alnæs 2012, Alnæs et al. 2014

FEniCS provides automated form assembly over finite element meshes and numerical linear algebra

```
from dolfin import *

# Define viscosity
def nu(u, T):
  return exp(-10.0*T)*dot(u, u)

# Define element spaces
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V * Q

# Define functions
T = Expression("...")
w = Function(W)
(u, p) = split(w)
(v, q) = TestFunctions(W)

# Define equation
F = (2*nu(u, T)*inner(eps(u), eps(v))
     + div(v)*p + div(u)*q
     + Ra*T*v[1])*dx
bcs = ...

# Solve F = 0 w.r.t w
solve(F == 0, w, bcs)
```



Automated assembly



High performance linear algebra

Logg and Wells, 2010; Hake, Logg and Wells 2012

Automated goal-oriented error control

What is goal-oriented error control?

The mathematician's viewpoint

Input

- PDE: find $u \in V$ such that

$$a(u, v) = L(v) \quad \text{or} \quad F(u; v) = 0 \qquad \forall\, v \in \hat{V}$$

- Quantity of interest/Goal: $\mathcal{M} : V \to \mathbb{R}$
- Tolerance: $\epsilon > 0$

Challenge

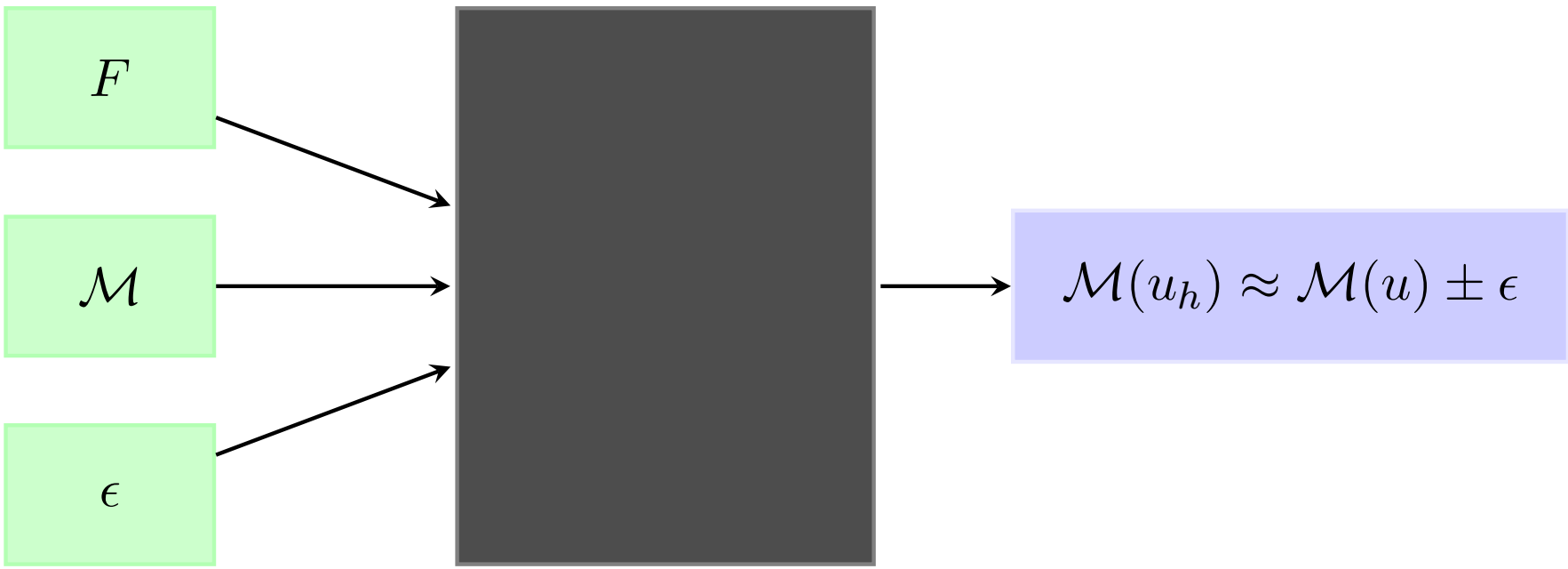
Find $V_h \subset V$ such that $|\mathcal{M}(u) - \mathcal{M}(u_h)| < \epsilon$ where $u_h \in V_h$ is determined by

$$a(u_h, v) = L(v) \quad \text{or} \quad F(u_h; v) = 0 \qquad \forall\, v \in \hat{V}_h$$

FEniCS provides automated goal-oriented error control for stationary variational problems

Find $U_h \subset U, V_h \subset V$ such that $|\mathcal{M}(u) - \mathcal{M}(u_h)| < \epsilon$ where $u_h \in U_h$ solves

$$F(u_h; v) = 0 \quad \forall\, v \in V_h$$




```
import dolfin
solve(F == 0, u, bc, tol=..., M=...)
```

Our approach mimics, generalizes and automates the classical, manual approach

$$|\mathcal{M}(u) - \mathcal{M}(u_h)| \approx -F(u_h; z) = r(z)$$

1. Compute adjoint

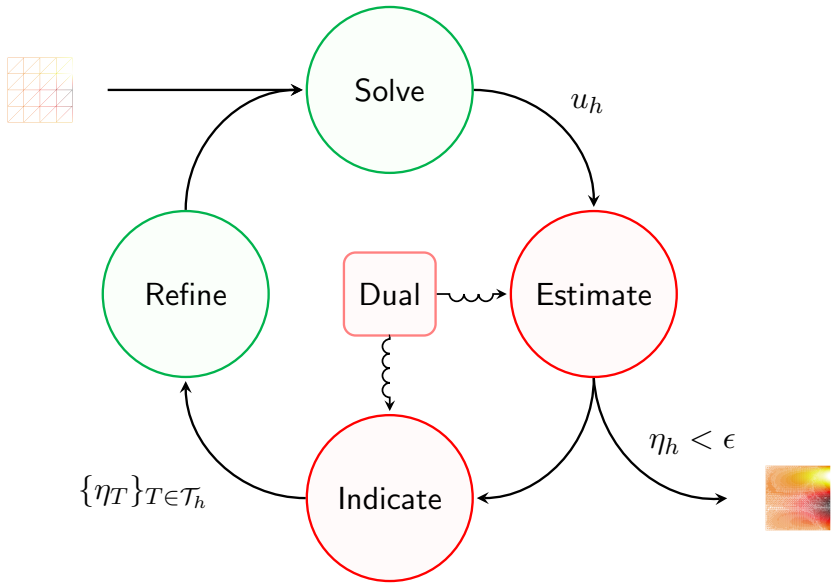
   $$F'^*(u_h; z_h, v) = M'(u_h; v) \quad \forall\, v \in V_h$$
   $$z_h \mapsto \tilde{z}_h$$

2. Automatically derive and compute residual form

   $$r(v) = \sum_T \langle R_T, v\rangle + \langle R_{\partial T}, v\rangle_{\partial T}$$

3. Evaluate error estimators

   $$\eta_h = r(\tilde{z}_h)$$
   $$\eta_T = |\langle R_T, w\rangle_T + \langle [R_{\partial T}], w\rangle_{\partial T}|$$
   $$w = \tilde{z}_h - \Pi_h \tilde{z}_h$$



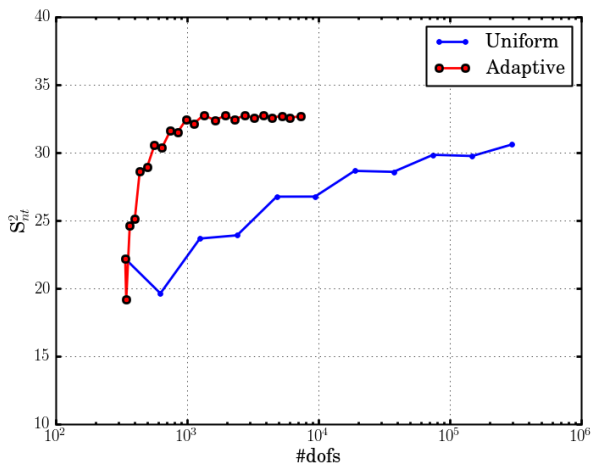
Note

Automated form manipulations and finite element operations are crucial to each step

Goal-oriented adaptivity gives significantly higher accuracy at significantly lower cost

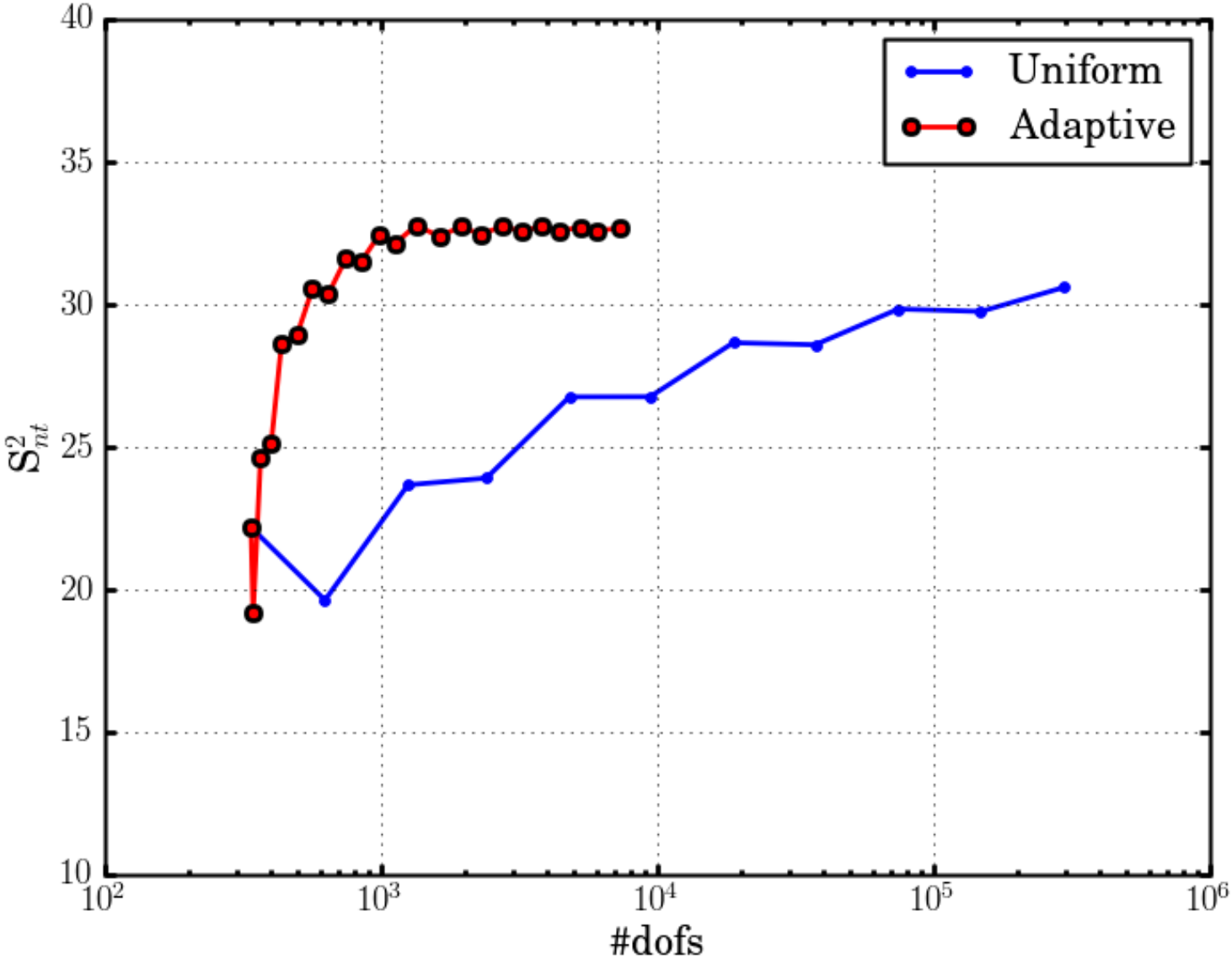
$$F(v;x) = \langle S(x), \mathrm{grad}(v)\rangle, \quad S = \frac{\partial W}{\partial F}, \quad W = \phi_I \mu_E \left(\frac{1}{2}(\|F\|^2 - \|I\|^2) + \beta^{-1}((\det F)^{-\beta} - 1)\right) + (\det F)\,(a\phi \ln\phi + b(1-\phi)\ln(1-\phi) + c\phi(1-\phi) + c_{FH})$$

$$\mathcal{M}(x) = \int_\Gamma S(x)^2_{\mathrm{nt}} \mathrm{dX}$$




```
# Mesh and function space
mesh = UnitSquare(12, 12)
V = VectorFunctionSpace(mesh, "CG", 1)

# Deformation
deformation = Expression(("x[0]", "x[1]"))
position = interpolate(deformation, V)

# Deformation gradient and its Jacobian
F = grad(position)
F = variable(F)
detF = det(F)

# Volume fraction definitions
phi_I = 0.8            # Reference volume fraction
phi = phi_I*inv(detF) # Volume fraction

# Elastic  potential
W_E = 1.0/2*((inner(F, F) - 2) + (detF**(-2) - 1))

# Flory-Huggins parameters and potential
a = 4.28001624e-05; b = 0.0428001624; c = 0.010354878
W_FH = a*phi*ln(phi) + b*(1 - phi)*ln(1-phi) + c*phi*(1-phi)

# Total potential
c_FH = 0.01338703463; scale = 1.e3
W = scale*(phi_I*W_E + detF*(W_FH + c_FH))

# Define stress-tensor
S = diff(W, F)

# Define variational form
v = TestFunction(V)
B = inner(S, grad(v))*dx

# Define goal (square shear stress)
M = S[0][1]*S[0][1]*ds(0)

solve(B == 0, position, bc, tol=0.1, M=M)
```

Goal-oriented adaptivity gives significantly higher accuracy at significantly lower cost



The Dolfin-adjoint Project

Adjoint solutions are key ingredients for pde-constrained optimization, sensitivity analysis, error control, ...

As an example, consider the optimal control problem:

$$\max_m J(u,m) \quad \text{while} \quad F(u,m) = 0$$

Gradient-based optimization algorithms require the gradient of J with respect to m.

$$\frac{\mathrm{d}J}{\mathrm{d}m} = J_u \frac{\partial u}{\partial m} + J_m$$

Define the adjoint solution z solving

$$F_u^* z = J_u$$

Then, the derivative computation only involves one forward solve for u and one backward solve for z independent of $\#m$:

$$\frac{\mathrm{d}J}{\mathrm{d}m} = -F_m z + J_m$$

Adjoints are highly relevant but adjoint models (code) are considered difficult to develop

[T]he automatic generation of optimal (in terms of robustness and efficiency) adjoint versions of large-scale simulation code is one of the great open challenges in the field of High-Performance Scientific Computing.

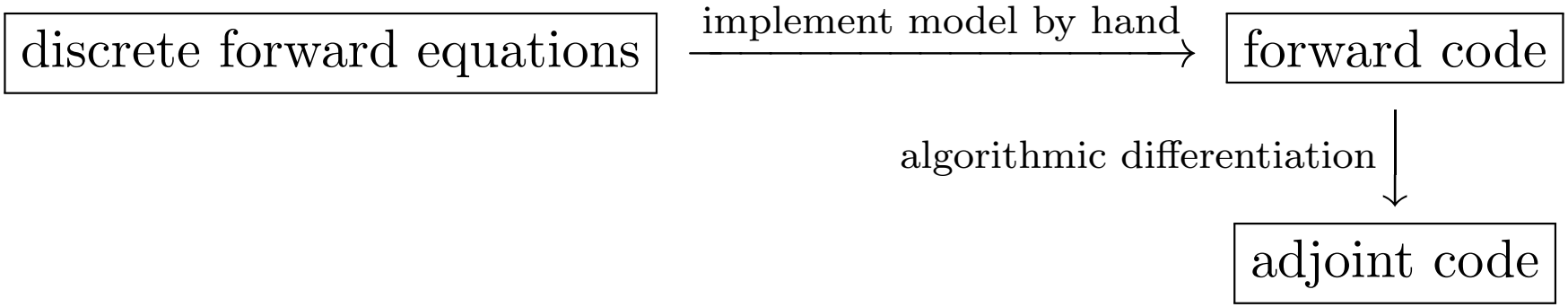
[Naumann: The Art of Differentiating Computer Programs, 2011]

Considering the importance of design to .. all of engineering, it is perhaps surprising that the development of adjoint codes has not been more rapid .. [I]t seems likely that part of the reason is its complexity.

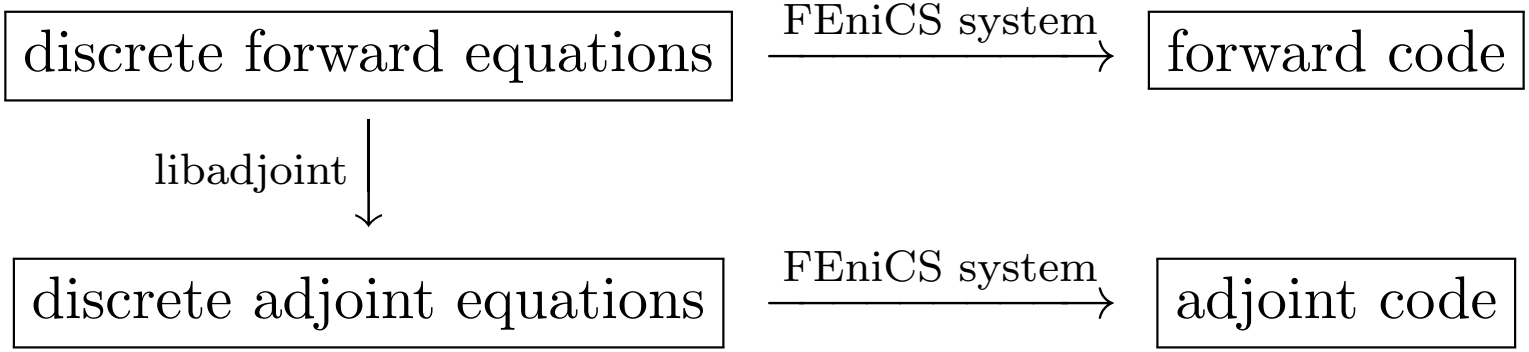
[Giles and Pierce: An Introduction to the Adjoint Approach to Design, 2000]

A new approach to adjoint model development

Traditional approach

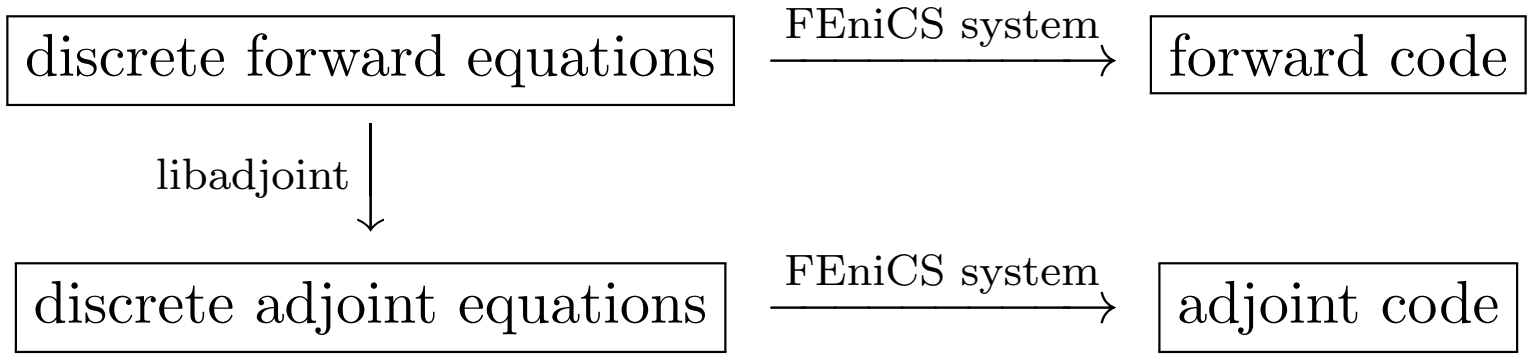


The dolfin-adjoint approach



[Farrell, Ham, Funke, R., Automated derivation of the adjoint of high-level transient FE programs, SISC, 2013]

A novel approach to developing adjoint models for forward models implemented in FEniCS



Cons/Pros

- The problem must be representable in the high-level language
- It does not apply to legacy code.
- The adjoint derivation is totally automatic ($3 - 10$ lines)
- The adjoint is efficient
- The adjoint works naturally in parallel (MPI and OpenMP) and can use checkpointing

[Farrell, Ham, Funke, R.: Automated derivation of the adjoint of high-level transient finite element programs, SISC, 2013]

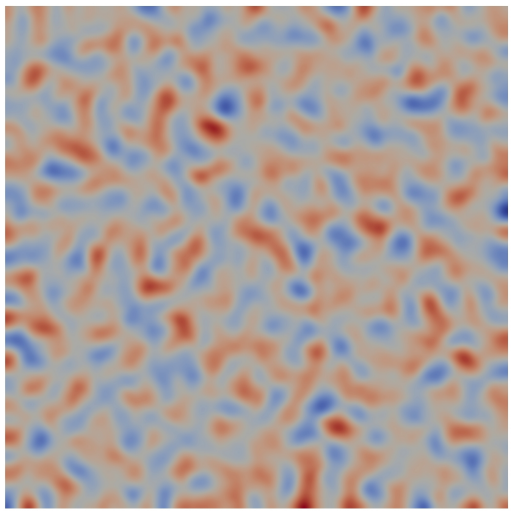
A (nontrivial) example: the Cahn-Hilliard equation

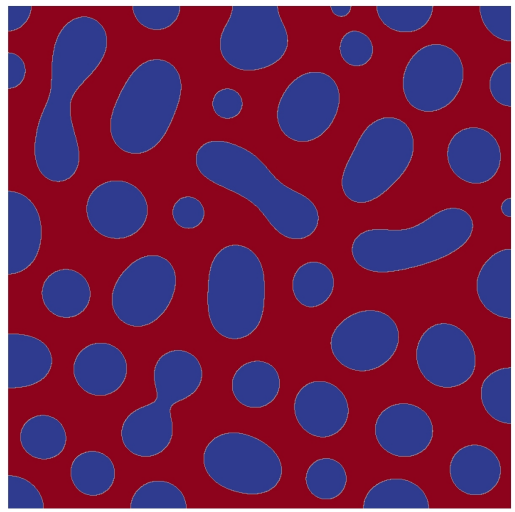
Given an initial concentration c_0, find the concentration field c such that

$$\frac{\partial c}{\partial t} - \nabla \cdot M\nabla\left(\frac{df}{dc} - \epsilon^2\nabla^2 c\right) = 0 \quad \text{in } \Omega,$$

$$M\nabla\left(\frac{df}{dc} - \epsilon^2\nabla^2 c\right) = 0, \quad M\epsilon^2\nabla c \cdot \hat{n} = 0 \quad \text{on } \partial\Omega,$$

$$f = 100c^2(1-c)^2, \quad c(t=0) = c_0 \quad \text{on } \Omega.$$





This approach to deriving and running adjoint models is automated, efficient and verifiable

Cahn-Hilliard equation with the Willmore functional

$$J(c(t), \mu(t)) = \frac{1}{4\epsilon} \int_{t=0}^{t=T} \int_{\Omega} \left(-\frac{1}{\epsilon} \mu(t) \right)^2 \mathrm{d}x \, \mathrm{d}t,$$

The adjoint computation is efficient

	Runtime (s)	Ratio
Forward model	103.93	
Forward model + annotation	104.24	1.002
Forward model + annotation + adjoint model	127.07	1.22

... also routines available for easy Taylor testing to verify correctness of gradient.

dolfin-adjoint
automatic adjoint models for FEniCS

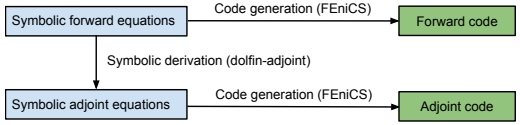
P. E. Farrell, S. W. Funke, D. A. Ham, M. E. Rognes
Simon@simula.no

The dolfin-adjoint project automatically derives and solves adjoint and tangent linear equations from high-level mathematical specifications of finite element discretizations of partial differential equations.

About dolfin-adjoint

How it works

Adjoint and tangent linear models form the basis of many numerical techniques such as sensitivity analysis, optimization, and stability analysis. However, the derivation and implementation of adjoint models for nonlinear or time-dependent models are notoriously challenging: the manual approach is time-consuming and error-prone and traditional automatic differentiation tools lack robustness and performance.
dolfin-adjoint solves this problem by automatically analyzing the high-level mathematical structure inherent in finite element methods. It raises the traditional abstraction of algorithmic differentiation from the level of individual floating point operations to that of whole systems of differential equations. This approach delivers a number of advantages over the previous state-of-the-art: robust hands-off automation of adjoint model derivation; computational efficiency approaching the theoretical optimum; and native parallel support inherited from the forward model.



◁ **Figure**: By adding a few lines of code to an existing FEniCS model, dolfin-adjoint computes tangent linear and adjoint solutions, gradients and Hessian actions of arbitrary user-specified functionals, and uses these derivatives in combination with sophisticated optimization algorithms or to conduct stability analyses

The implementation of dolfin-adjoint is based on the finite-element framework FEniCS. When the user runs a FEniCS model, dolfin-adjoint records the dependencies and structure of the forward equations. The resulting execution graph stores a mathematical representation of the forward equations. By reasoning about this graph, dolfin-adjoint can linearize the equations to derive a symbolic representation of the discrete tangent linear equations, and reverse the propagation of information to derive the corresponding adjoint equations. By invoking the FEniCS automatic code generator on these equations, dolfin-adjoint obtains solutions of the tangent linear and adjoint models, and can use these to compute consistent first and second order functional derivatives. dolfin-adjoint also has preliminary support for the Firedrake project.

Performance

dolfin-adjoint runs naturally in parallel, and inherits the scalability and code optimizations of FEniCS. To verify this, we benchmarked the sensitivity analysis and generalized stability application examples.

Sensitivity analysis example

CPUs	1	2	4	Optimal
Forward runtime (s)	40.3	19.6	13.2	
Adjoint runtime (s)	39.1	19.3	12.5	
Adjoint/Forward ratio	0.97	0.99	0.95	1.00

Generalized stability example

CPUs	1	2	Optimal
Forward runtime (s)	92.4	55.0	
Adjoint runtime (s)	41.4	25.9	
Adjoint/Forward ratio	0.45	0.47	0.5

Tables: The sensitivity analysis example is linear, while the generalized stability analysis example is nonlinear and converges on average in 2 Newton-iteration per timestep. Hence the adjoint model is expected to be twice as fast as the forward model.

Checkpointing

The adjoint equations depend on the forward solutions. However, storing the entire forward trajectory is infeasible for large, time-dependent simulations. In this case, dolfin-adjoint can employ a binomial checkpointing strategy via the revolve library. When activated, dolfin-adjoint automatically saves state checkpoints and uses them to recompute missing forward states to trade off memory requirements and computational effort. This allows for solving adjoint equations even for large-scale simulations. For instance, 390 checkpoints allow simulations with 10^7 time-steps at a cost of a 3× slow-down.

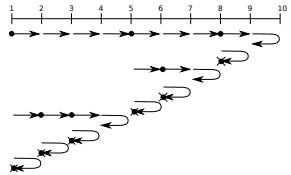


Figure: Visualisation of the optimal checkpointing strategy with 10 time levels and 3 checkpoints

Application examples

Sensitivity analysis

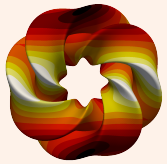
Consider the time dependent heat equation

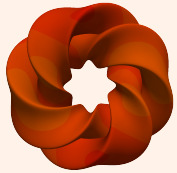
$$\frac{\partial u}{\partial t} - \nu\nabla^2 u = 0 \quad \text{in } \Omega \times (0, T),$$
$$u = g \quad \text{for } \Omega \times \{0\}.$$

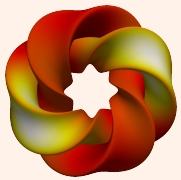
Here Ω is the Gray's Klein bottle, a closed 2D manifold embedded in 3D, T is the final time, u is the unknown temperature, ν is the thermal diffusivity, and g is the initial temperature.
The goal is to compute the sensitivity of the norm of temperature at the final time

$$J(u) = \int_\Omega u(t = T)^2$$

with respect to the initial temperature, that is $\mathrm{d}J/\mathrm{d}g$.


Initial temperature


Final temperature


Sensitivity

```
from dolfin import *
from dolfin_adjoint import *

# Solve the forward system
F = u*v*dx - u_old*v*dx +
    dt*nu*inner(grad(v),grad(u))*dx
while t <= T:
    t += dt
    solve(F == 0, u)

# Apply dolfin-adjoint
m = Control(g)
J = u**2*dx*dt[T]
dJdm = compute_gradient(J, m)
H = hessian(J, m)
```

△
Code: Implementation excerpt (the code including the complete forward model has 37 lines)

PDE-constrained optimization

This topology optimization example minimizes the compliance

$$\int_\Omega fT + \alpha \int_\Omega \nabla a \cdot \nabla a,$$

subject to the Poisson equation with mixed Dirichlet–Neumann conditions

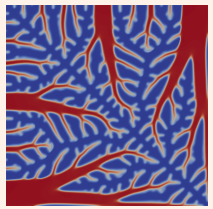
$$-\mathrm{div}(k(a)\nabla T) = f \quad \text{in } \Omega,$$
$$T = 0 \quad \text{on } \partial\Omega_D,$$
$$k(a)\nabla T = 0 \quad \text{on } \partial\Omega_N,$$

and additional control constraints

$$\int_\Omega a \le V \text{ and } 0 \le a(x) \le 1 \qquad \forall x \in \Omega.$$

Here Ω is the unit square, T is the temperature, a is the control ($a(x) = 1$ means material, $a(x) = 0$ means no material), f is a source term, $k(a)$ is the Solid Isotropic Material with Penalisation parameterization, α is a regularization term, and V is the volume bound on the control. Physically, the problem is to find the material distribution a that minimizes the integral of the temperature for a limited amount of conducting material.

```
from dolfin import *
from dolfin_adjoint import *
# ...
J = f*T*dx + alpha*inner(grad(a),grad(a))*dx
m = Control(a)
rf = ReducedFunctional(J, m)
minimize(rf, method="SLSQP", bounds=...)
```



△
Code: Implementation excerpt (the full code uses the IPOPT optimization package and has 56 lines)
◁ **Figure**: Optimal material distribution a for a unit square domain and $f = 10^{-2}$

Generalized stability analysis

This example performs a generalized stability analysis to find the perturbations to an initial condition that grow the most over some finite time. The governing equations are the two-dimensional vorticity-streamfunction formulation of the time-dependent Navier–Stokes equations, coupled to two advection equations for temperature and salinity: