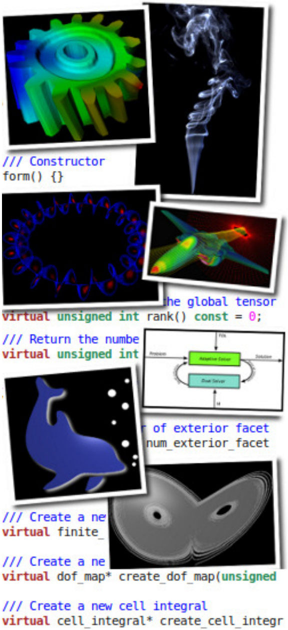
# Automated Solution of Differential Equations


Anders Logg

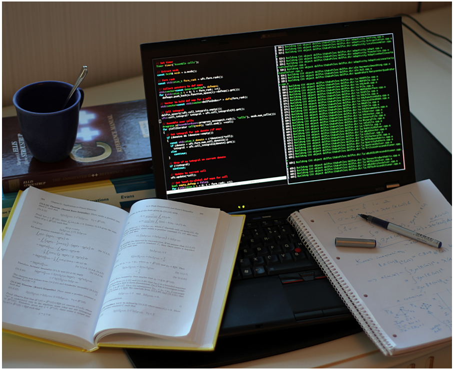
Mathematical Sciences
Chalmers University of Technology


Matematisk orientering TM1
8 december 2014



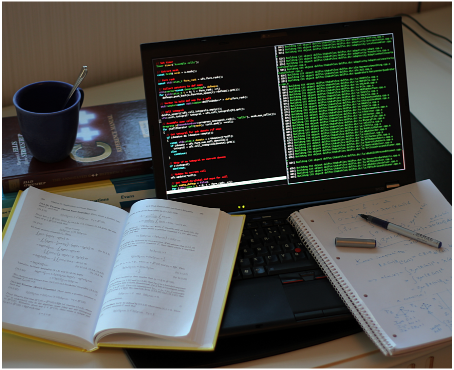
# How to solve differential equations

## (Using a numerical method)



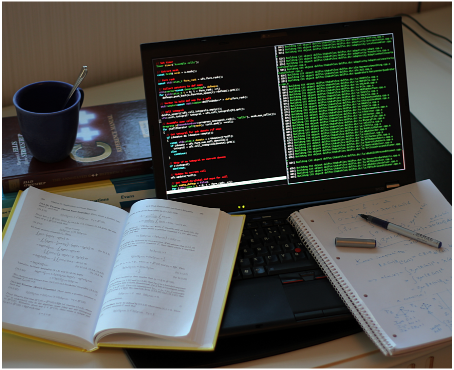
# How to solve differential equations

## (Using a numerical method)



- Understand the model

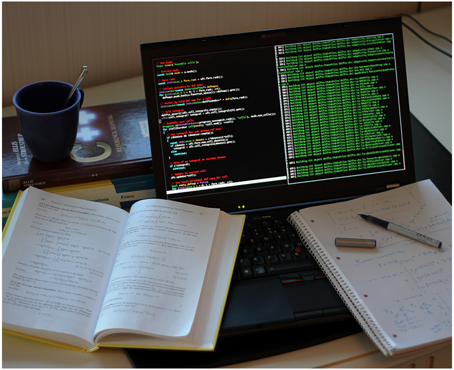
# How to solve differential equations
## (Using a numerical method)



- Understand the model
- Develop the numerical method

# How to solve differential equations

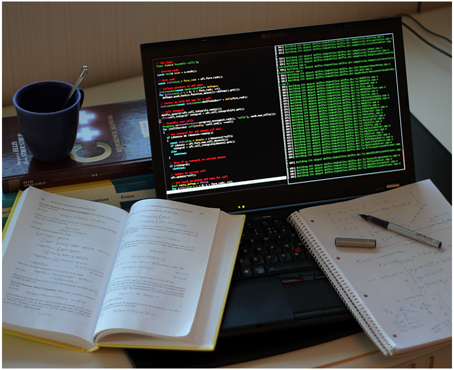
## (Using a numerical method)



- Understand the model
- Develop the numerical method
- Write the code

# How to solve differential equations

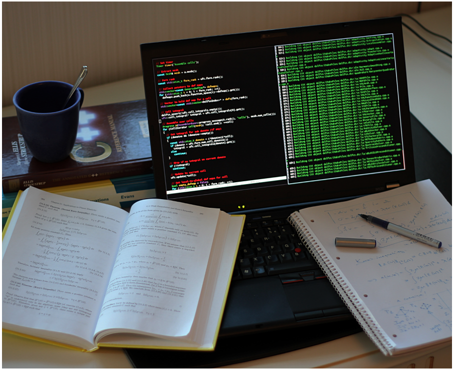
## (Using a numerical method)



- Understand the model
- Develop the numerical method
- Write the code
- Compile the code

# How to solve differential equations

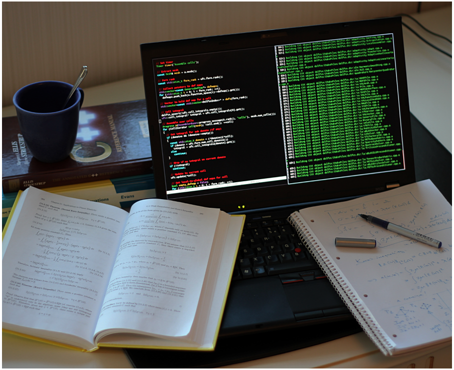
## (Using a numerical method)



- Understand the model
- Develop the numerical method
- Write the code
- Compile the code
- Run the code

# How to solve differential equations

## (Using a numerical method)



- Understand the model
- Develop the numerical method
- Write the code
- Compile the code
- Run the code

- **Solving a single equation can take years!**

# Let's look at an example (hyperelasticity)

$$F = I + \mathrm{grad}(u)$$
$$C = F^\top F$$
$$E = \frac{1}{2}(C - I)$$

$$W = \frac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2)$$

$$S = \frac{\partial W}{\partial E}$$
$$P = FS$$

$$\boxed{\begin{array}{rclr} -\mathrm{div}\,P & = & B & \text{in } \Omega \\ u & = & u_0 & \text{on } \partial\Omega \end{array}}$$

# Let's look at an example (hyperelasticity)

$$F = I + \mathrm{grad}(u)$$

$$C = F^\top F$$

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

**Strain measures**

$$W = \frac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2)$$

$$S = \frac{\partial W}{\partial E}$$

$$P = FS$$

$$\boxed{\begin{aligned} -\mathrm{div}\,P &= B && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}}$$

# Let's look at an example (hyperelasticity)

$$F = I + \mathrm{grad}(u)$$
$$C = F^\top F$$
$$E = \frac{1}{2}(C - I)$$

**Strain measures**

**Strain energy**

$$W = \frac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2)$$

$$S = \frac{\partial W}{\partial E}$$
$$P = FS$$

$$\boxed{\begin{array}{rcll} -\operatorname{div} P & = & B & \text{in } \Omega \\ u & = & u_0 & \text{on } \partial\Omega \end{array}}$$

# Let's look at an example (hyperelasticity)

$$F = I + \mathrm{grad}(u)$$

$$C = F^{\top} F$$

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

**Strain measures**

**Strain energy**

$$W = \frac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2)$$

**Stress tensors**

$$S = \frac{\partial W}{\partial E}$$

$$P = FS$$

$$\boxed{\begin{aligned} -\operatorname{div} P &= B && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}}$$

# Let's look at an example (hyperelasticity)

$$F = I + \mathrm{grad}(u)$$

$$C = F^\top F$$

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

- **Strain measures**

$$W = \frac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2)$$

- **Strain energy**

$$S = \frac{\partial W}{\partial E}$$

$$P = FS$$

- **Stress tensors**

$$\boxed{\begin{aligned} -\operatorname{div} P &= B && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}}$$

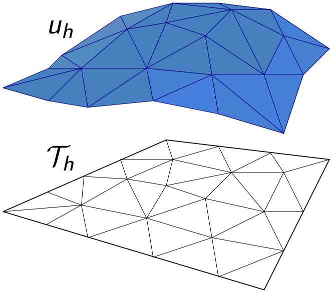
- **Partial differential equation**

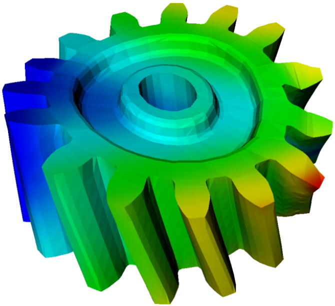
# The mathematical model is reflected in the finite element discretization

Find $u \in V_h \subset V$ such that

$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x = \int_\Omega B \cdot v \,\mathrm{d}x$$

for all $v \in \hat{V}_h \subset \hat{V}$



# But is obscured by the software implementation

```
subroutine quad(t,itype)
include 'commons'
integer t,itype,i,j,k,sub,qp
real x1,x2,x3,y1,y2,y3,
     det,area,
     dz1dx,dz2dx,dz3dx,
     dz1dy,dz2dy,dz3dy,
     qpx,qpy,
     db1dx,db2dx,db1dy,db2dy
real bcrhs,cu
real qpf,qpp,qpq,qpr
logical bndsid(3)

x1=xvert(vertex(1,t))
x2=xvert(vertex(2,t))
x3=xvert(vertex(3,t))
y1=yvert(vertex(1,t))
y2=yvert(vertex(2,t))
y3=yvert(vertex(3,t))

det = x1*(y2-y3) +
     x2*(y3-y1) + x3*(y1-y2)
area = abs(det/2.)

dz1dx = (y2-y3)/det
dz2dx = (y3-y1)/det
dz3dx = (y1-y2)/det
dz1dy = (x3-x2)/det
dz2dy = (x1-x3)/det
dz3dy = (x2-x1)/det
```

```
   do 10 qp=1,nqpt
      qpx = x1*quadpt(1,qp) +
           x2*quadpt(2,qp) + x3*quadpt(3,qp)
      qpy = y1*quadpt(1,qp) +
           y2*quadpt(2,qp) + y3*quadpt(3,qp)
      call pde(qpx,qpy,qpp,qpq,qpr,qpf)
      do 20 i=1,nadd
         db1dx = qpdbdz(1,row(i),qp)*dz1dx
               + qpdbdz(2,row(i),qp)*dz2dx
               + qpdbdz(3,row(i),qp)*dz3dx
         db1dy = qpdbdz(1,row(i),qp)*dz1dy
               + qpdbdz(2,row(i),qp)*dz2dy
               + qpdbdz(3,row(i),qp)*dz3dy
         db2dx = qpdbdz(1,col(i),qp)*dz1dx
               + qpdbdz(2,col(i),qp)*dz2dx
               + qpdbdz(3,col(i),qp)*dz3dx
         db2dy = qpdbdz(1,col(i),qp)*dz1dy
               + qpdbdz(2,col(i),qp)*dz2dy
               + qpdbdz(3,col(i),qp)*dz3dy
         add(i) = add(i) + quadw(qp)*
                 (qpp*db1dx*db2dx +
                  qpq*db1dy*db2dy +
                  qpr*qpbas(row(i),qp)*
                  qpbas(col(i),qp)
20    continue
      do 30 i=1,naddrs
         addrs(i) = addrs(i) + quadw(qp) *
              qpf * qpbas(rowrs(i),qp)
30    continue
10 continue
```

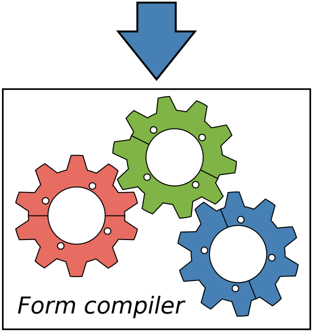
# A new idea

**R. C. Kirby, M. G. Knepley, A. Logg, L. R. Scott**
Optimizing the evaluation of finite element matrices
SIAM Journal on Scientific Computing 27 (2005)

**R. C. Kirby and A. Logg**
A compiler for variational forms
ACM Transactions on Mathematical Software 32 (2006)

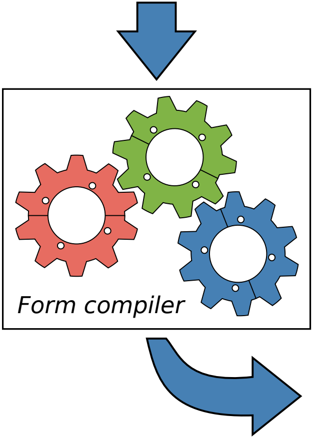
# The code can be generated

$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x = \int_\Omega B \cdot v \,\mathrm{d}x$$



# The code can be generated

$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x = \int_\Omega B \cdot v \,\mathrm{d}x$$




```
// Extract vertex coordinates
const double * const * x = c.coordinates;

// Compute Jacobian of affine map from reference cell
const double J_00 = x[1][0] - x[0][0];
const double J_01 = x[2][0] - x[0][0];
const double J_02 = x[3][0] - x[0][0];
const double J_10 = x[1][1] - x[0][1];
const double J_11 = x[2][1] - x[0][1];
const double J_12 = x[3][1] - x[0][1];
const double J_20 = x[1][2] - x[0][2];
const double J_21 = x[2][2] - x[0][2];
const double J_22 = x[3][2] - x[0][2];

// Compute sub determinants
const double d_00 = J_11*J_22 - J_12*J_21;
const double d_01 = J_12*J_20 - J_10*J_22;
const double d_02 = J_10*J_21 - J_11*J_20;
const double d_10 = J_02*J_21 - J_01*J_22;
const double d_11 = J_00*J_22 - J_02*J_20;
const double d_12 = J_01*J_20 - J_00*J_21;
const double d_20 = J_01*J_12 - J_02*J_11;
const double d_21 = J_02*J_10 - J_00*J_12;
const double d_22 = J_00*J_11 - J_01*J_10;

// Compute determinant of Jacobian
double detJ = J_00*d_00 + J_10*d_10 + J_20*d_20;

// Compute inverse of Jacobian
const double K_00 = d_00 / detJ;
... [7328 lines of code]
```

**This is 7328 lines of code**

**that no one had to write**

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \frac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \frac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \frac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)

C = F.T*F
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \tfrac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \frac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
W = lmda/2*(tr(E)**2) + mu*tr(E*E)
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \frac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
W = lmda/2*(tr(E)**2) + mu*tr(E*E)
S = diff(W, E)
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \tfrac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
W = lmda/2*(tr(E)**2) + mu*tr(E*E)
S = diff(W, E)
P = F*S
```

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^\top F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\,\mathrm{tr}(E^2) \\
S &= \tfrac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
W = lmda/2*(tr(E)**2) + mu*tr(E*E)
S = diff(W, E)
P = F*S
```

$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x - \int_\Omega B \cdot v \,\mathrm{d}x = 0$$

# And we recover the mathematical notation

$$
\begin{aligned}
F &= I + \mathrm{grad}(u) \\
C &= F^{\top} F \\
E &= \tfrac{1}{2}(C - I) \\
W &= \tfrac{\lambda}{2}(\mathrm{tr}(E))^2 + \mu\, \mathrm{tr}(E^2) \\
S &= \tfrac{\partial W}{\partial E} \\
P &= FS
\end{aligned}
$$

```
F = I + grad(u)
C = F.T*F
E = variable(0.5*(C - I))
W = lmda/2*(tr(E)**2) + mu*tr(E*E)
S = diff(W, E)
P = F*S
```

$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x - \int_\Omega B \cdot v \,\mathrm{d}x = 0$$

$$\Leftrightarrow$$

$$\mathtt{inner}(\mathtt{P}, \mathtt{grad}(\mathtt{v})) * \mathtt{dx} - \mathtt{dot}(\mathtt{B}, \mathtt{v}) * \mathtt{dx} == 0$$

**Can we completely automate the solution of differential equations?**

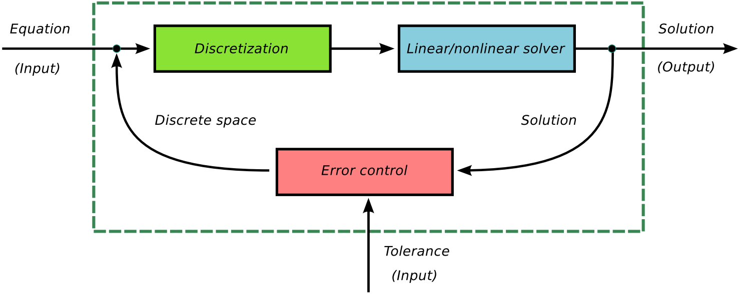
# Automated solution of differential equations

## Input

- $A(u) = f$
- $\epsilon > 0$

## Output

- $u_h \approx u$
- $\|u - u_h\| \leq \epsilon$



# Automated solution of differential equations

## Key steps

(i) Automated discretization ✓ (2006)

(ii) Automated error control ✓ (2010)

(iii) Automated generation of well-posed discretizations

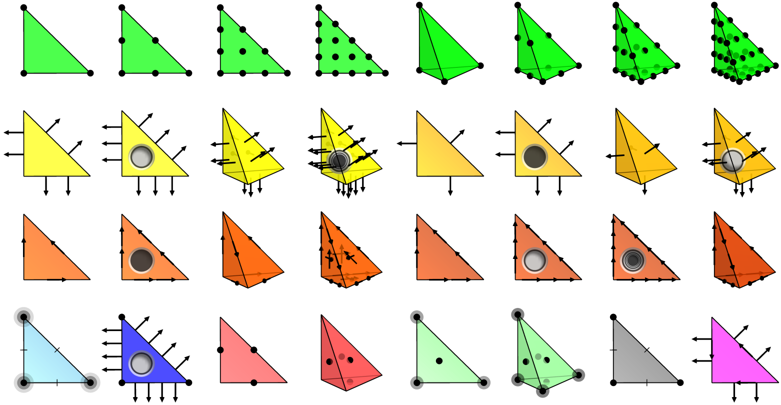
## Key techniques

- Adaptive finite element methods
- Automatic code generation

# (i) Automated discretization

# Automated discretization by automated FEM

- Automated generation of basis functions
- Automated evaluation of variational forms
- Automated finite element assembly



# The Finite Element Method (FEM)

## Differential equation (strong form)

$$-\Delta u = f$$

## Weak form

$$\text{Find } u \in V : \quad \underbrace{\langle \mathrm{grad}\, u, \mathrm{grad}\, v \rangle}_{a(u,v)} = \underbrace{\langle f, v \rangle}_{L(v)} \quad \forall\, v \in V$$

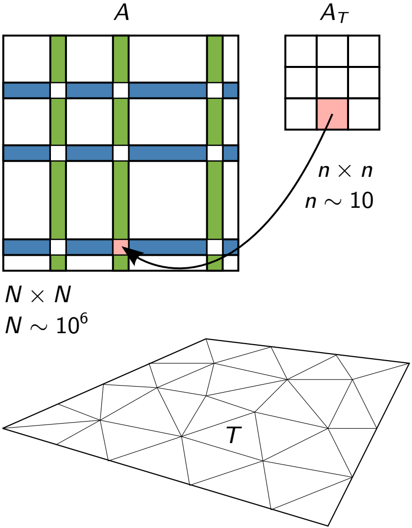
## Finite element method

$$\text{Find } u_h \in V_h : \quad \langle \mathrm{grad}\, u_h, \mathrm{grad}\, v \rangle = \langle f, v \rangle \quad \forall\, v \in V_h$$

## Solution algorithm (for $u_h = \sum_{j=1}^{N} U_j \phi_j$)

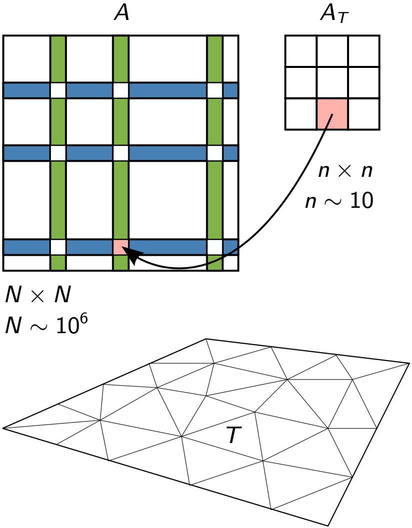
$$AU = b \qquad A_{ij} = \langle \mathrm{grad}\, \phi_j, \mathrm{grad}\, \phi_i \rangle \qquad b_i = \langle f, \phi_i \rangle$$

# Finite element assembly




- Compute $A_{ij} = a(\phi_j, \phi_i)$

# Finite element assembly

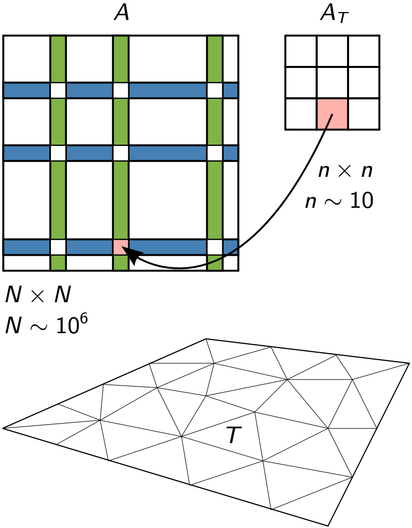



- Compute $A_{ij} = a(\phi_j, \phi_i)$
- Compute the element matrix

$$A_{T,ij} = a_T(\phi_j, \phi_i)$$

on each triangle $T$

# Finite element assembly

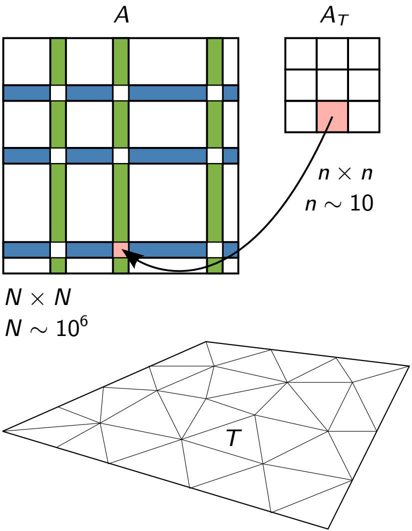



- Compute $A_{ij} = a(\phi_j, \phi_i)$
- Compute the element matrix

$$A_{T,ij} = a_T(\phi_j, \phi_i)$$

  on each triangle $T$
- Assemble $\{A_T\}_T$ into the global matrix $A$

# Finite element assembly

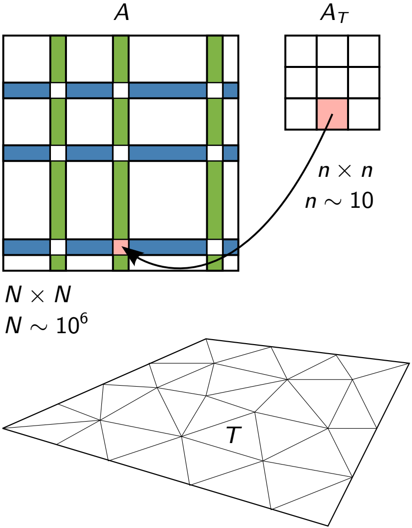



- Compute $A_{ij} = a(\phi_j, \phi_i)$
- Compute the element matrix

$$A_{T,ij} = a_T(\phi_j, \phi_i)$$

  on each triangle $T$
- Assemble $\{A_T\}_T$ into the global matrix $A$
- Generate code for the evaluation of $A_T$

# Finite element assembly




- Compute $A_{ij} = a(\phi_j, \phi_i)$
- Compute the element matrix

$$A_{T,ij} = a_T(\phi_j, \phi_i)$$

  on each triangle $T$
- Assemble $\{A_T\}_T$ into the global matrix $A$

- Generate code for the evaluation of $A_T$
- Use the structure of $A_T$ to generate *optimized* code

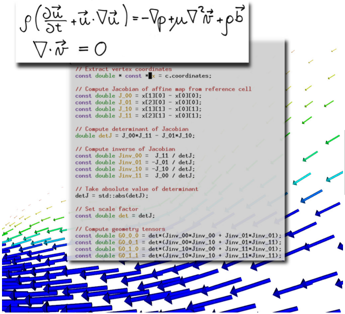
# Automatic code generation

## Input

Equation (variational problem)

## Output

Efficient application-specific code




*Equation (variational form)*

↓

**Form compiler**

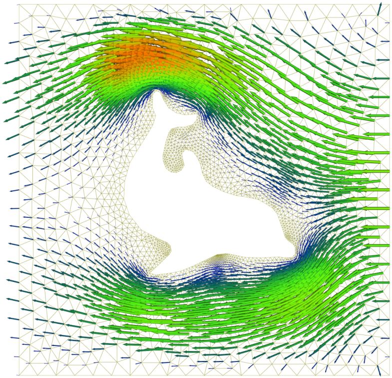
↓

*Application-specific code*

# Example: the Stokes equations

## Differential equation

$$
\begin{aligned}
-\operatorname{div}\operatorname{grad} u + \operatorname{grad} p &= f \\
\operatorname{div} u &= 0
\end{aligned}
$$



# Example: the Stokes equations

## Weak form

Find $(u, p) \in V \times Q$ such that

$$a((u,p),(q,v)) = L((v,q)) \quad \forall\,(v,q) \in \hat{V} \times \hat{Q}$$

where

$$a((u,p),(q,v)) = \int_\Omega \mathrm{grad}\, u : \mathrm{grad}\, v - p\,\mathrm{div}\, v + \mathrm{div}\, u\, q \,\mathrm{d}x$$

$$L((v,q)) = \int_\Omega fv \,\mathrm{d}x$$

# Example: the Stokes equations

## Implementation

*Python code*

```
P2 = VectorElement("Lagrange", triangle, 2)
P1 = FiniteElement("Lagrange", triangle, 1)
TH = P2 * P1

(u, p) = TrialFunctions(TH)
(v, q) = TestFunctions(TH)

f = Coefficient(P2)

a = (inner(grad(u), grad(v))-p*div(v)+div(u)*q)*dx
L = dot(f, v)*dx
```

# Example: the Stokes equations

## Implementation

*Python code*

```
P2 = VectorElement("Lagrange", triangle, 2)
P1 = FiniteElement("Lagrange", triangle, 1)
TH = P2 * P1

(u, p) = TrialFunctions(TH)
(v, q) = TestFunctions(TH)

f = Coefficient(P2)

a = (inner(grad(u), grad(v))-p*div(v)+div(u)*q)*dx
L = dot(f, v)*dx
```

$$a = \int_\Omega \mathrm{grad}\, u : \mathrm{grad}\, v - p \,\mathrm{div}\, v + \mathrm{div}\, u \, q \,\mathrm{d}x$$

## (ii) Automated error control

# Automated goal-oriented error control

## Input

- Variational problem: Find $u \in V$: $a(u, v) = L(v) \quad \forall\, v \in V$
- Quantity of interest: $\mathcal{M} : V \to \mathbb{R}$
- Tolerance: $\epsilon > 0$

## Objective

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

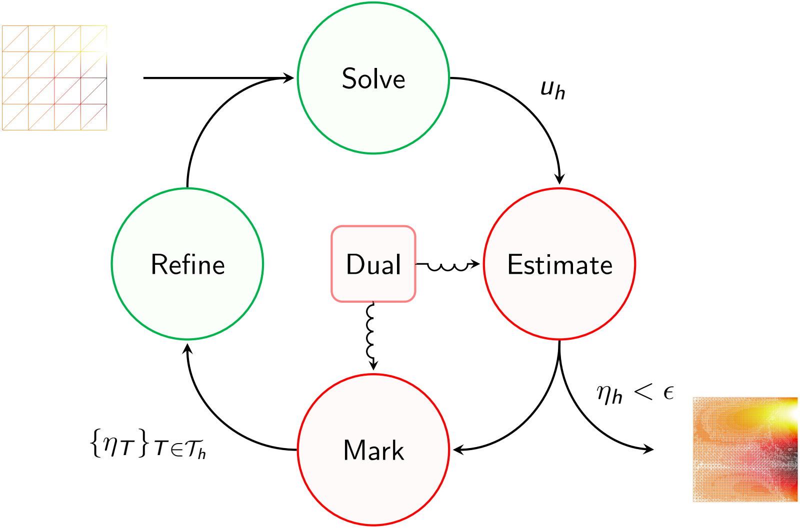
$$a(u_h, v) = L(v) \quad \forall\, v \in V_h$$

## Automated in FEniCS (for linear and nonlinear PDE)

*Python code*

```
solve(a == L, u, M=M, tol=1e-3)
```

# Adaptivity = solve – estimate – mark – refine



# Error analysis

Define (weak) residual:

$$r(v) = L(v) - a(u_h, v)$$

Introduce dual problem:

$$\text{Find } z \in V\colon \quad a^*(z, v) = \mathcal{M}(v) \quad \forall\, v \in V$$

Error representation:

$$\begin{aligned}
\mathcal{M}(u) - \mathcal{M}(u_h) &= \mathcal{M}(e) \\
&= a^*(z, e) \\
&= a(e, z) \\
&= a(u, z) - a(u_h, z) \\
&= L(z) - a(u_h, z) \\
&\equiv r(z)
\end{aligned}$$

# A posteriori error estimate for Poisson

$$a(u, v) = \int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v \, \mathrm{d}x \quad L(v) = \int_\Omega fv \, \mathrm{d}x$$

Recall error representation:

$$\mathcal{M}(u) - \mathcal{M}(u_h) = r(z) = \int_\Omega fz - \operatorname{grad} u_h \cdot \operatorname{grad} z \, \mathrm{d}x$$

Residual decomposition:

$$r(v) = \sum_{T \in \mathcal{T}_h} \int_T \underbrace{(f + \Delta u_h)}_{R_T} v \, \mathrm{d}x - \int_{\partial T} \underbrace{\operatorname{grad} u_h \cdot n}_{R_{\partial T}} v \, \mathrm{d}s$$

Error indicators:

$$\eta_T = |\langle R_T, z - \pi_h z \rangle_T + \langle [\![ R_{\partial T} ]\!], z - \pi_h z \rangle_{\partial T}|$$

# Key steps to automated error control

- Automated linearization
- Automated generation of the dual problem
- Automated integration by parts:

$$r_T(v) = \int_T R_T \cdot v \,\mathrm{d}x + \int_{\partial T} R_{\partial T} \cdot v \,\mathrm{d}s$$

  Test against bubble functions to solve for $R_T$ and $R_{\partial T}$
- Automated computation of error indicators:

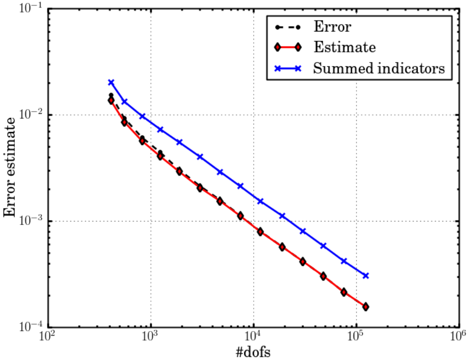
$$\eta_T = |\langle R_T, \tilde{z}_h - z_h \rangle_T + \langle [\![ R_{\partial T} ]\!], \tilde{z}_h - z_h \rangle_{\partial T}|$$

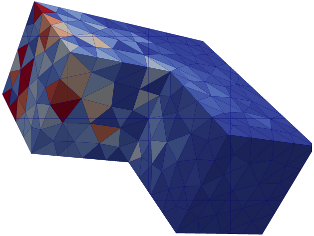
- Automated mesh refinement
- Dual problem solved on same function space and extrapolated

---

Rognes, Logg, *Automated Goal-Oriented Error Control I: Stationary Variational Problems* (2013)

# Poisson's equation

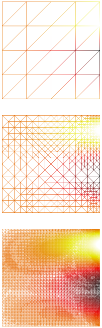


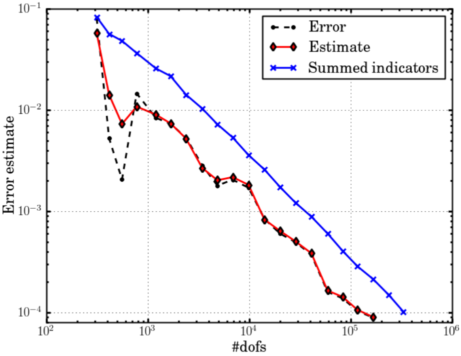



$$a(u, v) = \langle \mathrm{grad}\, u, \mathrm{grad}\, v \rangle$$

$$\mathcal{M}(u) = \int_\Gamma u \,\mathrm{d}s, \quad \Gamma \subset \partial\Omega$$

# A three-field mixed elasticity formulation

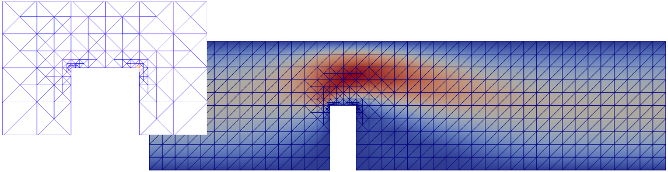





$$a((\sigma, u, \gamma), (\tau, v, \eta)) = \langle A\sigma, \tau\rangle + \langle u, \operatorname{div}\tau\rangle + \langle \operatorname{div}\sigma, v\rangle + \langle\gamma, \tau\rangle + \langle\sigma, \eta\rangle$$

$$\mathcal{M}((\sigma, u, \eta)) = \int_\Gamma g\,\sigma\cdot n\cdot t\,\mathrm{d}s$$

# Incompressible Navier–Stokes



Outflux $\approx 0.4087 \pm 10^{-4}$

Uniform

1.000.000 dofs, $N$ hours

Adaptive

5.200 dofs, 127 seconds

*Python code*

```python
from dolfin import *

class Noslip(SubDomain): ...

mesh = Mesh("channel-with-flap.xml.gz")
V = VectorFunctionSpace(mesh, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = V*Q

# Define test functions and unknown(s)
(v, q) = TestFunctions(W)
w = Function(W)
(u, p) = split(w)

# Define (non-linear) form
n = FacetNormal(mesh)
p0 = Expression("(4.0 - x[0])/4.0")
F = (0.02*inner(grad(u), grad(v)) + inner(grad(u)*u), v)*dx
    - p*div(v) + div(u)*q + dot(v, n)*p0*ds

# Define goal functional
M = u[0]*ds(0)

# Compute solution
tol = 1e-4
solve(F == 0, w, bcs, M, tol)
```

[Rognes, Logg, *Automated Goal-Oriented Error Control I* (2013)]

# The FEniCS Project

# FEniCS is an automated programming environment for differential equations

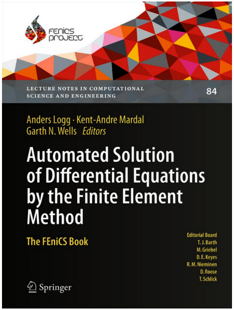
- C++/Python FEM library
- Initiated 2003 in Chicago
- 1000–2000 monthly downloads
- Part of Debian and Ubuntu
- Licensed under the GNU LGPL

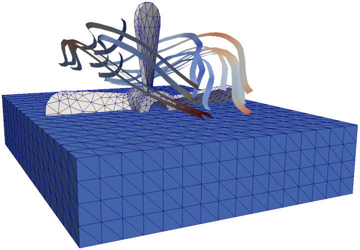
`http://fenicsproject.org/`

### Collaborators (in order of appearance)

*Chalmers University of Technology, University of Chicago, KTH Royal Institute of Technology, Simula Research Laboratory, University of Cambridge, Texas Tech University, University of Texas at Austin, Baylor University*

# Key features

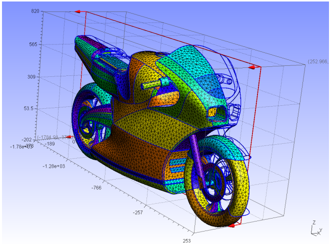


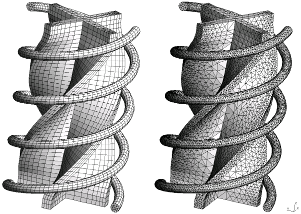



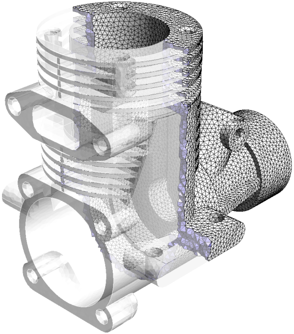
- Automated discretization
- Automated error control
- General families of finite elements, including arbitrary order continuous and discontinuous Lagrange elements, BDM, RT, Nédélec, ...
- General mixed elements
- Distributed (clusters) and shared memory (multicore) parallelism
- High performance linear algebra (PETSc, Trilinos)
- Mesh generation, adaptive mesh refinement
- Extensive documentation

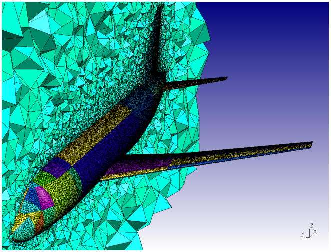
# Multimesh FEM for multiphysics

# Challenge 1: Mesh generation

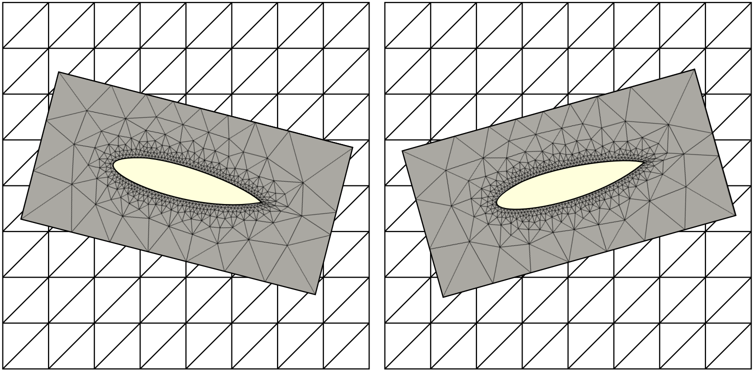


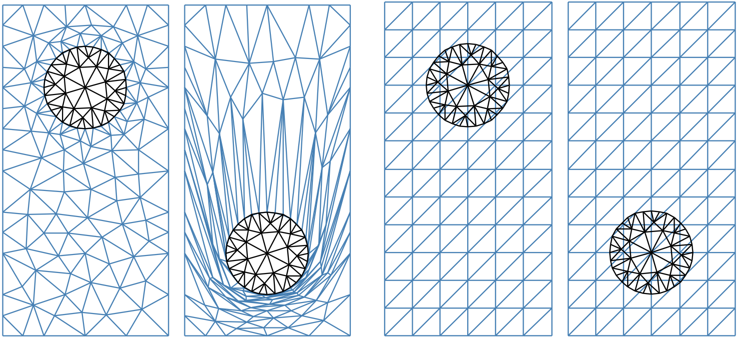




# Challenge 2: Geometry-dependent parameter studies



# Challenge 3: Evolving geometries



# Multimesh finite element methods

- Galerkin framework based on Nitsche's method
- Arbitrary combination of overlapping and intersecting meshes
- Proper method design leads to a theoretical basis including
  - Inf-sup stability
  - Optimal a priori error estimates
  - Optimal algebraic condition number estimates

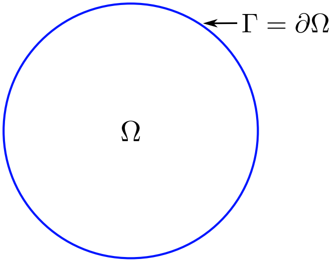
  Theoretical program by Burman, Hansbo, Hansbo, Larson
- Efficient and robust implementation in 3D
- Application to fluid flow and FSI
- Distributed as free/open-source software as part of FEniCS

Collaborators: Larson, Massing, Rognes, Johansson, Lundholm

# Nitsche's method for Dirichlet boundary conditions

## Differential equation

$$
\begin{aligned}
-\Delta u &= f \quad \text{in } \Omega \\
u &= g \quad \text{on } \Gamma
\end{aligned}
$$




## Weak form

Find $u \in V_h$ s.t.

$a(u, v) = L(v) \, \forall \, v \in V_h$

$$
a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \underbrace{\int_\Gamma \nabla u \cdot \mathbf{n} v \, \mathrm{d}S}_{\text{Consistency}} - \underbrace{\int_\Gamma \nabla v \cdot \mathbf{n} u \, \mathrm{d}S}_{\text{Symmetrization}} + \underbrace{\gamma \int_\Gamma h^{-1} u v \, \mathrm{d}S}_{\text{Penalization}}
$$

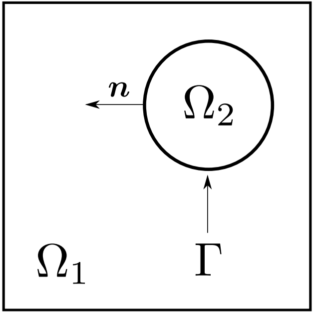
$$
L(v) = \int_\Omega f v \, \mathrm{d}x - \underbrace{\int_\Gamma \nabla v \cdot \mathbf{n} g \, \mathrm{d}S}_{\text{Symmetrization}} + \underbrace{\gamma \int_\Gamma h^{-1} g v \, \mathrm{d}S}_{\text{Penalization}}
$$

# Nitsche's method for interface conditions

## Domain decomposition

Find $(u_1, u_2)$ such that

$$
\begin{aligned}
-\Delta u_i &= f_i \quad \text{in } \Omega_i, \quad i = 1, 2 \\
[\nabla u \cdot \mathbf{n}] &= 0 \quad \text{on } \Gamma \\
[u] &= 0 \quad \text{on } \Gamma
\end{aligned}
$$




## Variational formulation

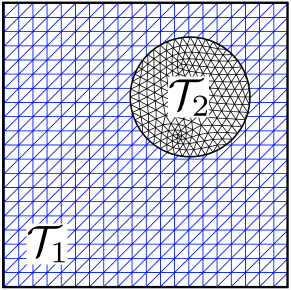
$$
a(u, v) = \sum_{i=1,2} \int_{\Omega_i} \nabla u \cdot \nabla v \,\mathrm{d}x
- \underbrace{\int_\Gamma \langle \nabla u \cdot \mathbf{n} \rangle [v] \,\mathrm{d}S}_{\text{Consistency}}
- \underbrace{\int_\Gamma \langle \nabla v \cdot \mathbf{n} \rangle [u] \,\mathrm{d}S}_{\text{Symmetrization}}
+ \underbrace{\gamma \int_\Gamma h^{-1} [u][v] \,\mathrm{d}S}_{\text{Penalty/Stabilization}}
$$

$$
L(v) = \int_\Omega f v \,\mathrm{d}x
$$

# Multimesh FEM for Stokes

## Interface formulation

$$
\begin{aligned}
-\Delta \mathbf{u}_i + \nabla p_i &= \mathbf{f}_i \quad \text{in } \Omega_i, \quad i = 1,2 \\
\mathbf{n} \cdot \mathbf{u}_i &= 0 \quad \text{in } \Omega_i, \quad i = 1,2 \\
[\mathbf{u}] &= 0 \quad \text{on } \Gamma \\
[\partial_{\mathbf{n}} \mathbf{u} - p\mathbf{n}] &= 0 \quad \text{on } \Gamma \\
\mathbf{u} &= 0 \quad \text{on } \partial\Omega
\end{aligned}
$$




## Finite element method

$$
\begin{aligned}
a_h(\mathbf{u}_h, \mathbf{v}_h) = (\nabla \mathbf{u}_h, \nabla \mathbf{v}_h)_{\Omega_1 \cup \Omega_2} &- (\langle \partial_{\mathbf{n}} \mathbf{u}_h \rangle, [\mathbf{v}_h])_\Gamma \\
&- (\langle \partial_{\mathbf{n}} \mathbf{v}_h \rangle, [\mathbf{u}_h])_\Gamma + \gamma (h^{-1} [\mathbf{u}_h], [\mathbf{v}_h])_\Gamma
\end{aligned}
$$

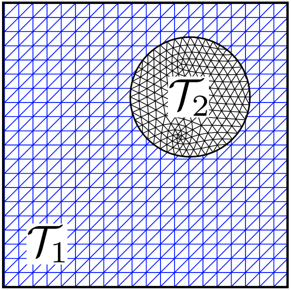
$$
b_h(\mathbf{v}_h, q_h) = -(\nabla \cdot \mathbf{v}_h, q_h)_{\Omega_1 \cup \Omega_2} + (\mathbf{n} \cdot [\mathbf{v}_h], \langle q_h \rangle)_\Gamma
$$

$$
S_h(\mathbf{u}_h, p_h; \mathbf{v}_h, q_h) = \underbrace{\delta \sum_{i=1,2} \sum_{T \in \mathcal{T}_i \cap \Omega_i} h_T^2 (-\Delta \mathbf{u}_h + \nabla p_h, -\alpha \Delta \mathbf{v}_h + \beta \nabla q_h)_T}_{\text{Stabilization}}
$$

# Multimesh FEM for Stokes

## Interface formulation

$$
\begin{aligned}
-\Delta \mathbf{u}_i + \nabla p_i &= \mathbf{f}_i \quad \text{in } \Omega_i, \quad i = 1,2 \\
\mathbf{n} \cdot \mathbf{u}_i &= 0 \quad \text{in } \Omega_i, \quad i = 1,2 \\
[\mathbf{u}] &= 0 \quad \text{on } \Gamma \\
[\partial_{\mathbf{n}} \mathbf{u} - p\mathbf{n}] &= 0 \quad \text{on } \Gamma \\
\mathbf{u} &= 0 \quad \text{on } \partial\Omega
\end{aligned}
$$




## Finite element method

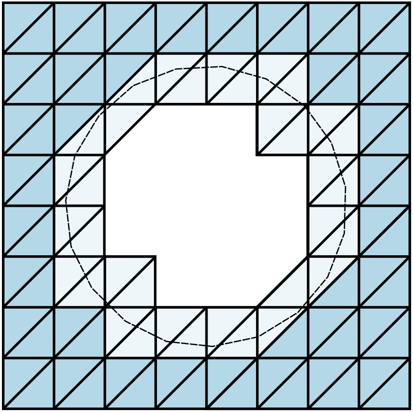
$$
\begin{aligned}
a_h(\mathbf{u}_h, \mathbf{v}_h) = {} & (\nabla \mathbf{u}_h, \nabla \mathbf{v}_h)_{\Omega_1 \cup \Omega_2} - (\langle \partial_{\mathbf{n}} \mathbf{u}_h \rangle, [\mathbf{v}_h])_\Gamma \\
& - (\langle \partial_{\mathbf{n}} \mathbf{v}_h \rangle, [\mathbf{u}_h])_\Gamma + \gamma (h^{-1}[\mathbf{u}_h], [\mathbf{v}_h])_\Gamma
\end{aligned}
$$

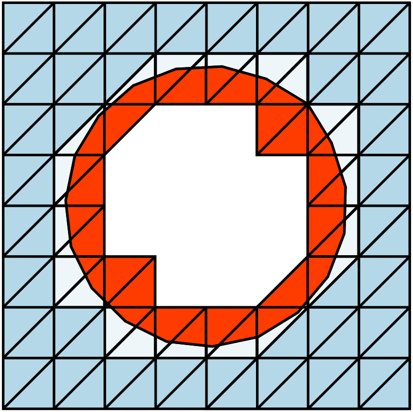
$$
b_h(\mathbf{v}_h, q_h) = -(\nabla \cdot \mathbf{v}_h, q_h)_{\Omega_1 \cup \Omega_2} + (\mathbf{n} \cdot [\mathbf{v}_h], \langle q_h \rangle)_\Gamma
$$

$$
S_h(\mathbf{u}_h, p_h; \mathbf{v}_h, q_h) = \underbrace{\delta \sum_{T \in \mathcal{T}_1^* \cup \mathcal{T}_2} h_T^2 (-\Delta \mathbf{u}_h + \nabla p_h, -\alpha \Delta \mathbf{v}_h + \beta \nabla q_h)_T}_{\text{Stabilization and ghost penalty } p}
$$

$$
s_h(\mathbf{u}_h, \mathbf{v}_h) = \underbrace{(\nabla(\mathbf{u}_{h,1} - \mathbf{u}_{h,2}), \nabla(\mathbf{v}_{h,1} - \mathbf{v}_{h,1}))_{\Omega_O}}_{\text{Ghost penalty for } u}
$$

# Ghost penalties are added in the interface zone





$$\delta \sum_{T \in \mathcal{T}_1^* \cup \mathcal{T}_2} h_T^2(-\Delta \mathbf{u}_h + \nabla p_h, -\alpha \Delta \mathbf{v}_h + \beta \nabla q_h)_T$$

$$(\nabla(\mathbf{u}_{h,1} - \mathbf{u}_{h,2}), \nabla(\mathbf{v}_{h,1} - \mathbf{v}_{h,1}))_{\Omega_O}$$

Ghost penalty for $p$

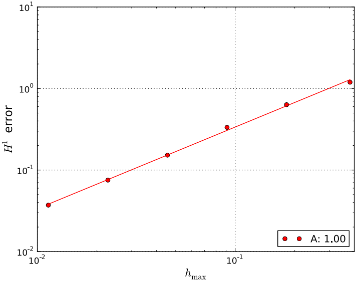
Ghost penalty for $u$

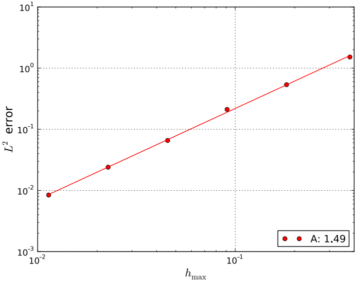
# Optimal a priori error estimate

**Theorem**

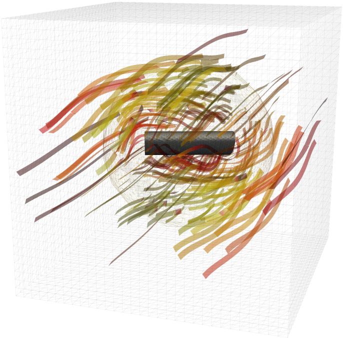
*Let $k, l \geqslant 1$ and assume that $(\mathbf{u}, p) \in [H^{k+1}(\Omega)]^d \times H^{l+1}(\Omega)$ is a (weak) solution of the Stokes problem. Then the finite element solution $(\mathbf{u}_h, p_h) \in V_h^k \times Q_h^l$ satisfies the following error estimate:*

$$|||(\mathbf{u} - \mathbf{u}_h, p - p_h)||| \lesssim h^k |\mathbf{u}|_{k+1,\Omega} + h^{l+1} |p|_{l+1,\Omega}.$$

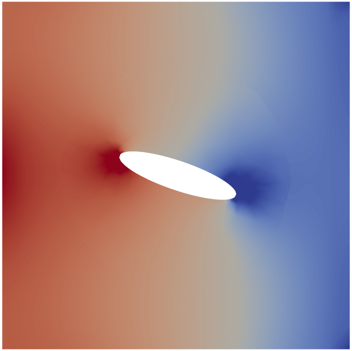


# Application: Stokes flow around an airfoil

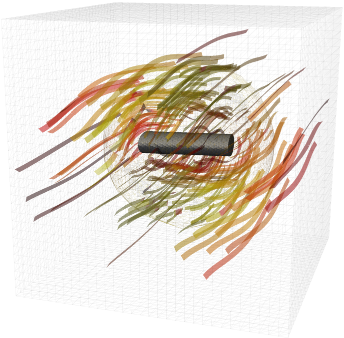


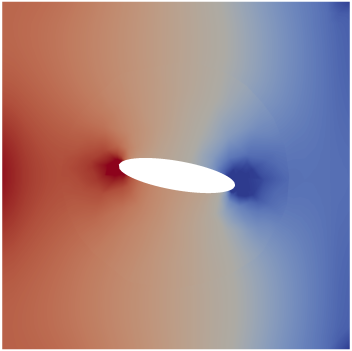
Velocity streamlines



Pressure

# Application: Stokes flow around an airfoil

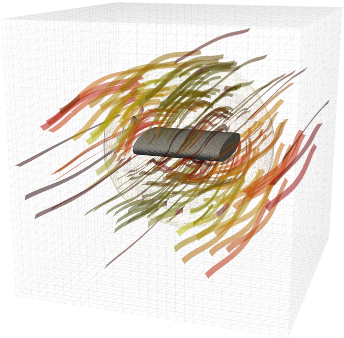




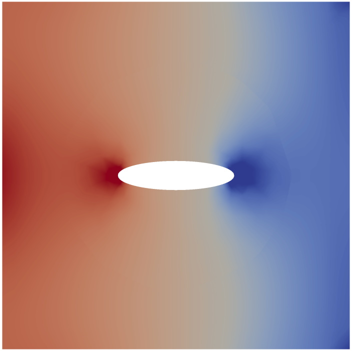
Velocity streamlines

Pressure

# Application: Stokes flow around an airfoil

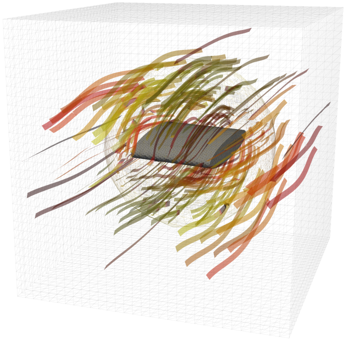


Velocity streamlines

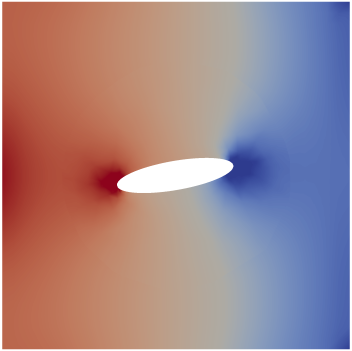


Pressure

# Application: Stokes flow around an airfoil

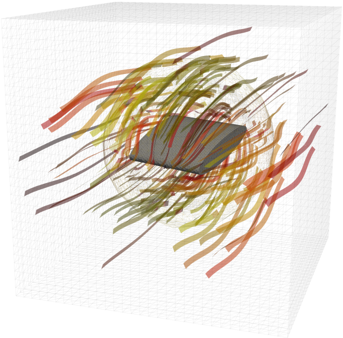


Velocity streamlines

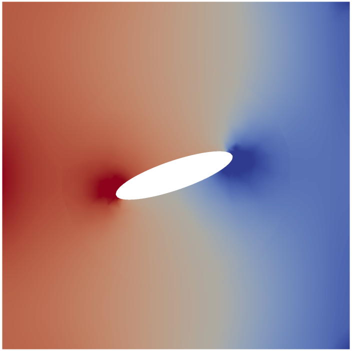


Pressure

# Application: Stokes flow around an airfoil



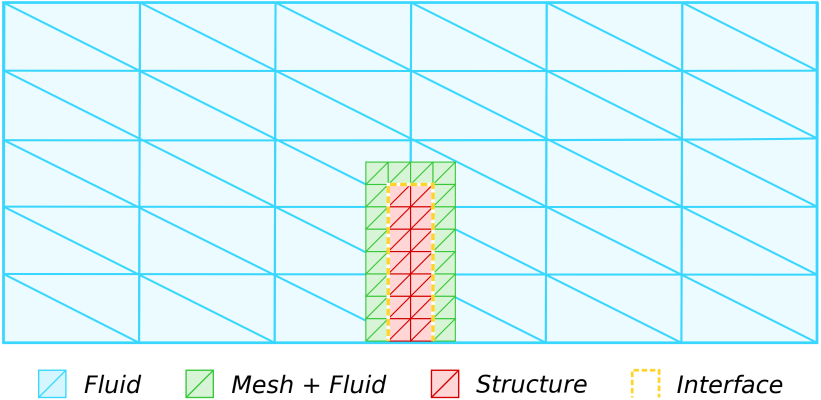
Velocity streamlines



Pressure

# Fluid–structure interaction on cut meshes

# Multimesh FEM for FSI



# A stationary FSI problem: state equations

*Fluid (F)*

$$\boxed{\begin{aligned} \nabla \cdot \sigma_F(u_F, p_F) &= 0 \\ \nabla \cdot u_F &= 0 \end{aligned}}$$

$$\boxed{\nabla \cdot \widetilde{\sigma}_M(\widetilde{u}_M) = 0}$$

*Mesh (M)*

$$\boxed{\nabla \cdot \widehat{\Pi}(\widehat{u}_S) = 0}$$

*Structure (S)*

# A stationary FSI problem: state equations

*Fluid (F)*

$$\boxed{\begin{aligned} \nabla \cdot \sigma_F(u_F, p_F) &= 0 \\ \nabla \cdot u_F &= 0 \end{aligned}}$$

*Interface (I)*

$$\boxed{\begin{aligned} \widetilde{u}_M &= \widehat{u}_S \\ u_F &= 0 \\ \widehat{\Pi}(\widehat{u}_S) \cdot \widehat{n}_S &= \widetilde{J}_M \widetilde{\sigma}(\widetilde{u}_F, \widetilde{p}_F) \widetilde{F}_M^{-T} \widehat{n}_S \end{aligned}}$$

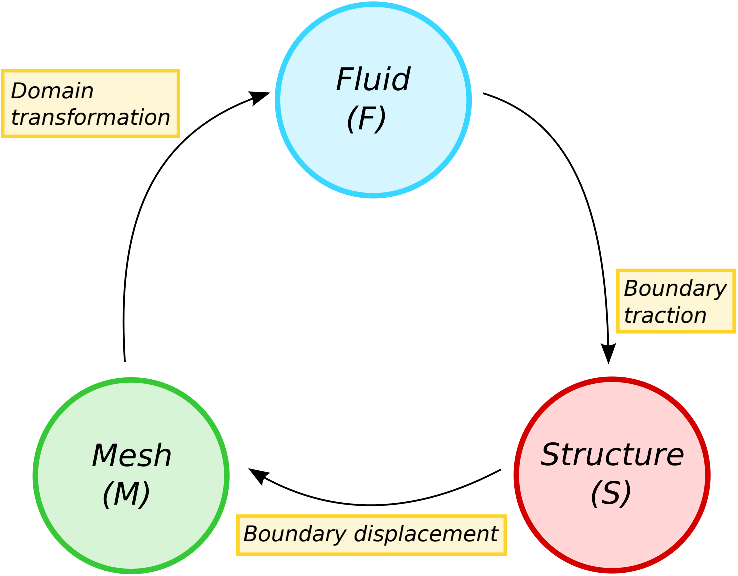
$$\boxed{\nabla \cdot \widetilde{\sigma}_M(\widetilde{u}_M) = 0}$$

*Mesh (M)*

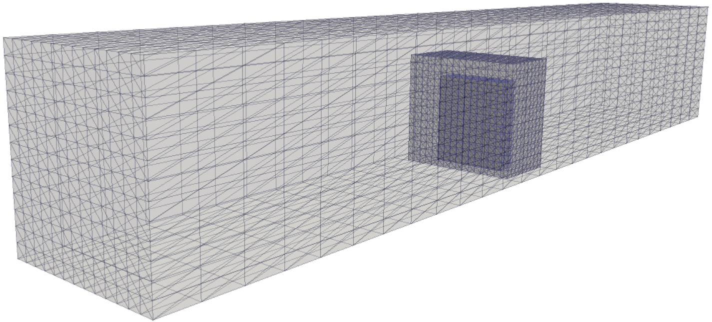
$$\boxed{\nabla \cdot \widehat{\Pi}(\widehat{u}_S) = 0}$$

*Structure (S)*

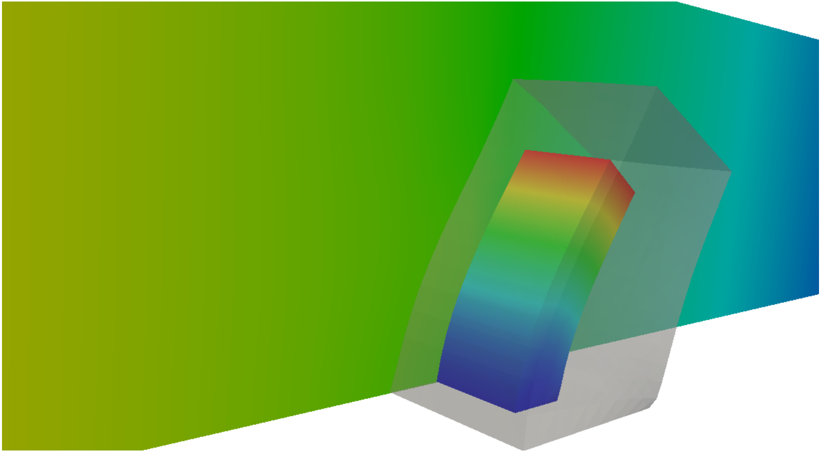
# The stationary FSI problem can be solved by a classical fixed-point method



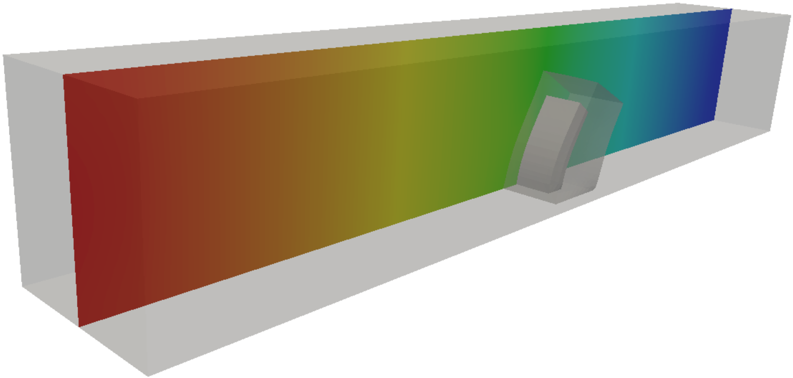
# Application: elastic flap in channel



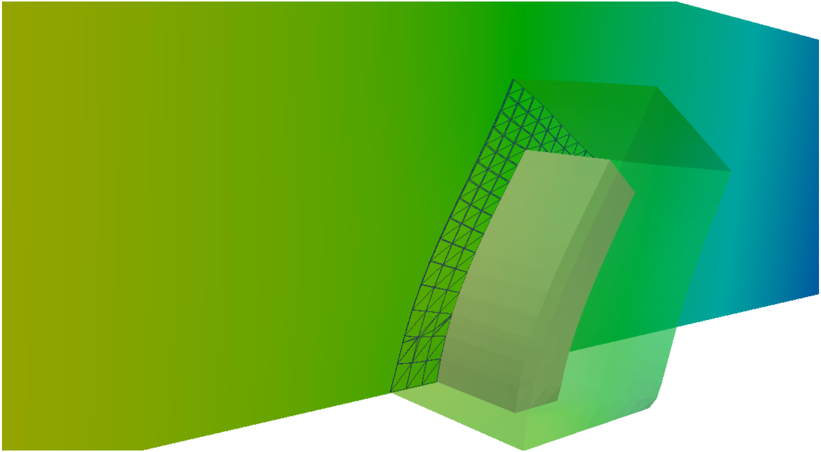
# Flap in channel: Displacement



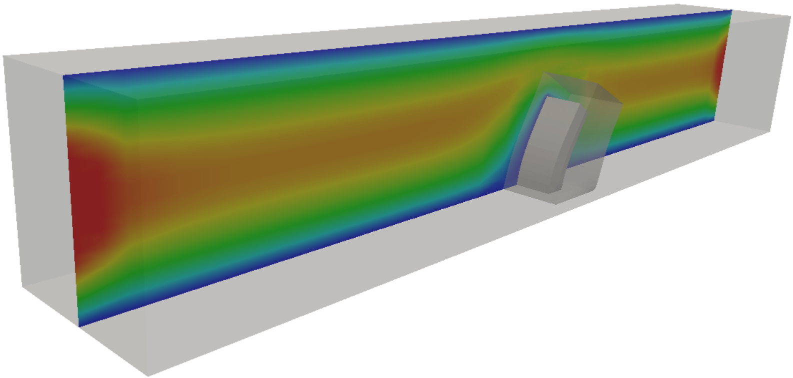
# Flap in channel: Pressure



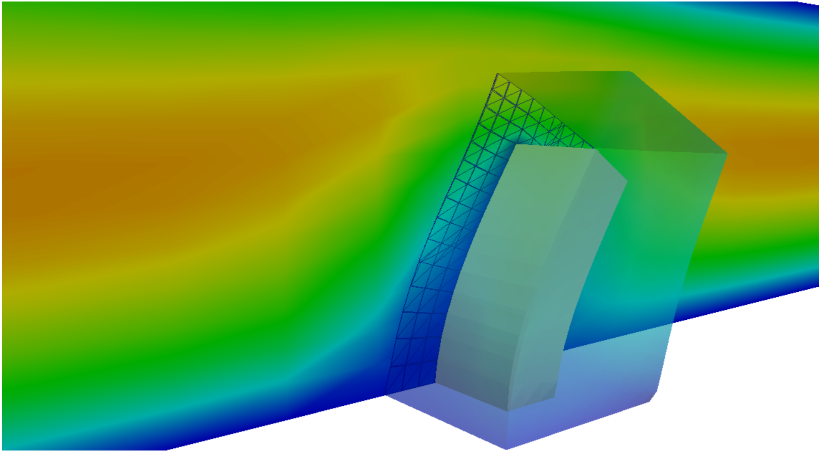
# Flap in channel: Pressure



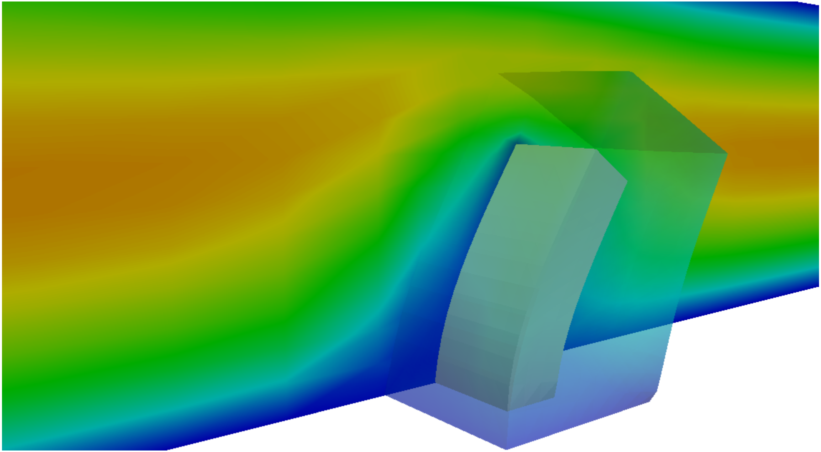
# Flap in channel: Velocity



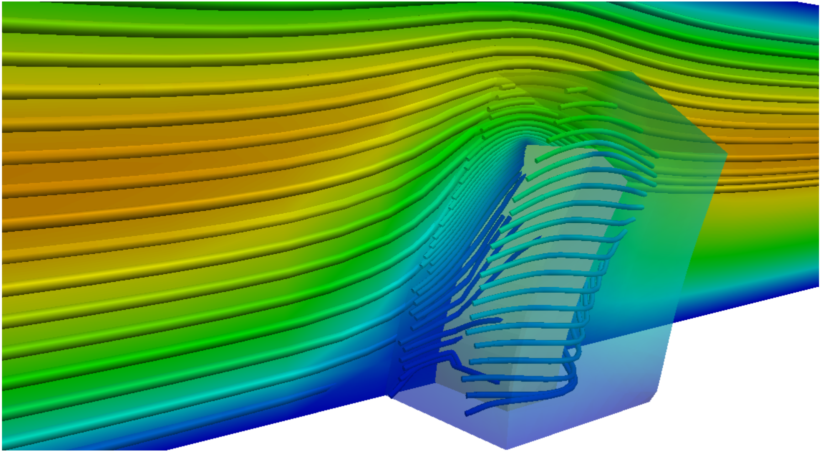
# Flap in channel: Velocity



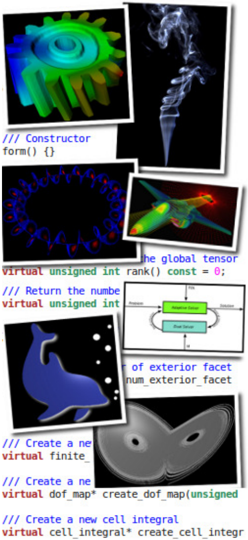
# Flap in channel: Velocity



# Flap in channel: Velocity



# Summary




- The solution of PDEs can be **automated**
- Based on **automatic code generation**
- New tools for:
  - Application scientists and engineers
  - Numerical analysts
- New possibilities for:
  - Mathematical modeling
  - Teaching of PDE and FEM

`http://fenicsproject.org/`