The FEniCS Project

Presented by Anders Logg*
Simula Research Laboratory

PDESoft 2012, Münster
2012–06–20

* Credits: `http://fenicsproject.org/about/team.html`

What is FEniCS?

FEniCS is an automated programming environment for differential equations

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

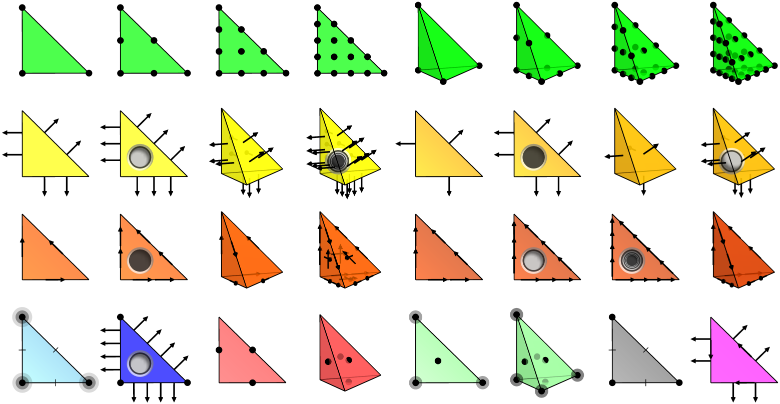
`http://fenicsproject.org/`

Collaborators

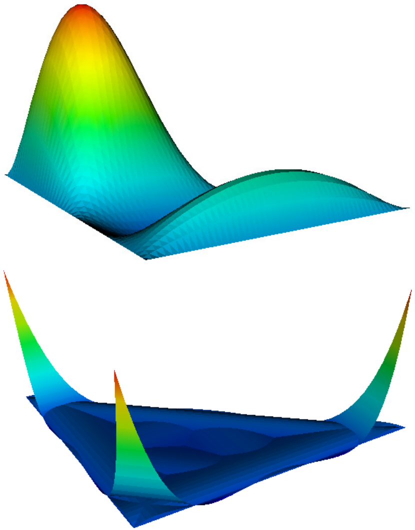
Simula Research Laboratory, University of Cambridge, University of Chicago, Texas Tech University, University of Texas at Austin, KTH Royal Institute of Technology, . . .

FEniCS is automated FEM

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



Finite element basis functions



- CG_q (P_q)
- DG_q
- BDM_q
- BDFM_q
- RT_q
- Nedelec 1st/2nd kind
- Crouzeix–Raviart
- Morley
- Hermite
- Argyris
- Bell

  ...
- $\mathcal{P}_q\Lambda^k$, $\mathcal{P}_q^-\Lambda^k$

How to use FEniCS?

Installation

- Official packages for Debian and Ubuntu
- Drag and drop installation on Mac OS X
- Binary installer for Windows
- Automated installation from source

Hello World in FEniCS: problem formulation

Poisson's equation

$$
\begin{aligned}
-\Delta u &= f \quad \text{in } \Omega \\
u &= 0 \quad \text{on } \partial\Omega
\end{aligned}
$$

Finite element formulation

Find $u \in V$ such that

$$
\underbrace{\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x}_{a(u,v)} = \underbrace{\int_\Omega f \, v \, \mathrm{d}x}_{L(v)} \quad \forall \, v \in V
$$

Hello World in FEniCS: implementation

```
from dolfin import *

mesh = UnitSquare(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]*x[1]")

a = dot(grad(u), grad(v))*dx
L = f*v*dx

bc = DirichletBC(V, 0.0, DomainBoundary())

u = Function(V)
solve(a == L, u, bc)
plot(u)
```

Linear elasticity

Differential equation

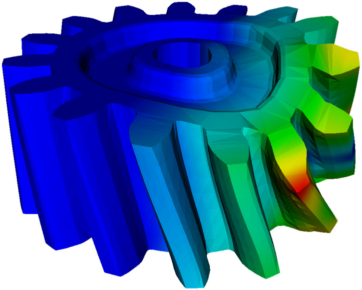
Differential equation:

$$-\nabla \cdot \sigma(u) = f$$

where

$$\sigma(v) = 2\mu\epsilon(v) + \lambda \mathrm{tr}\,\epsilon(v)\,I$$
$$\epsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^\top)$$

- Displacement $u = u(x)$
- Stress $\sigma = \sigma(x)$



Linear elasticity

Variational formulation

Find $u \in V$ such that

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

where

$$\begin{aligned} a(u, v) &= \langle \sigma(u), \epsilon(v) \rangle \\ L(v) &= \langle f, v \rangle \end{aligned}$$

Linear elasticity

Implementation

```
element = VectorElement("Lagrange", "tetrahedron", 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

def epsilon(v):
    return 0.5*(grad(v) + grad(v).T)

def sigma(v):
    return 2.0*mu*epsilon(v) + lmbda*tr(epsilon(v))*I

a = inner(sigma(u), epsilon(v))*dx
L = dot(f, v)*dx
```

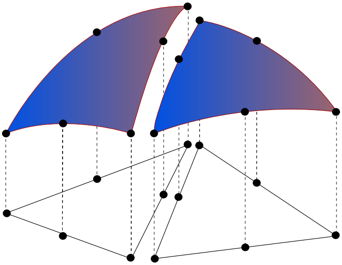
Poisson's equation with DG elements

Differential equation

Differential equation:

$$-\Delta u = f$$

- $u \in L^2$
- u discontinuous across element boundaries



Poisson's equation with DG elements

Variational formulation (interior penalty method)

Find $u \in V$ such that

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

where

$$\begin{aligned}
a(u, v) &= \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x \\
&\quad + \sum_S \int_S -\langle \nabla u \rangle \cdot [\![v]\!]_n - [\![u]\!]_n \cdot \langle \nabla v \rangle + (\alpha/h)[\![u]\!]_n \cdot [\![v]\!]_n \,\mathrm{d}S \\
&\quad + \int_{\partial\Omega} -\nabla u \cdot [\![v]\!]_n - [\![u]\!]_n \cdot \nabla v + (\gamma/h) u v \,\mathrm{d}s \\
L(v) &= \int_\Omega f v \,\mathrm{d}x + \int_{\partial\Omega} g v \,\mathrm{d}s
\end{aligned}$$

Poisson's equation with DG elements

Implementation

```
V = FunctionSpace(mesh, "DG", 1)

u = TrialFunction(V)
v = TestFunction(V)

f = Expression(...)
g = Expression(...)
n = FacetNormal(mesh)
h = CellSize(mesh)

a = dot(grad(u), grad(v))*dx
  - dot(avg(grad(u)), jump(v, n))*dS
  - dot(jump(u, n), avg(grad(v)))*dS
  + alpha/avg(h)*dot(jump(u, n), jump(v, n))*dS
  - dot(grad(u), jump(v, n))*ds
  - dot(jump(u, n), grad(v))*ds
  + gamma/h*u*v*ds
```

Oelgaard, Logg, Wells, *Automated Code Generation for Discontinuous Galerkin Methods* (2008)

Simple prototyping and development in Python

```python
# Tentative velocity step (sigma formulation)
U = 0.5*(u0 + u)
F1 = rho*(1/k)*inner(v, u - u0)*dx +
rho*inner(v, grad(u0)*(u0 - w))*dx \
   + inner(epsilon(v), sigma(U, p0))*dx \
   + inner(v, p0*n)*ds - mu*inner(grad(U).T*n, v)*ds \
   - inner(v, f)*dx
a1 = lhs(F1)
L1 = rhs(F1)
```

```python
class StVenantKirchhoff(MaterialModel):

    def model_info(self):
        self.num_parameters = 2
        self.kinematic_measure = \
          "GreenLagrangeStrain"

    def strain_energy(self, parameters):
        E = self.E
        [mu, lmbda] = parameters
        return lmbda/2*(tr(E)**2) + mu*tr(E*E)
```

```python
class GentThomas(MaterialModel):

    def model_info(self):
        self.num_parameters = 2
        self.kinematic_measure = \
          "CauchyGreenInvariants"

    def strain_energy(self, parameters):
        I1 = self.I1
        I2 = self.I2

        [C1, C2] = parameters
        return C1*(I1 - 3) + C2*ln(I2/3)
```

```python
# Time-stepping loop
while True:

    # Fixed point iteration on FSI problem
    for iter in range(maxiter):

        # Solve fluid subproblem
        F.step(dt)

        # Transfer fluid stresses to structure
        Sigma_F = F.compute_fluid_stress(u_F0, u_F1,
                                         p_F0, p_F1,
                                         U_M0, U_M1)
        S.update_fluid_stress(Sigma_F)

        # Solve structure subproblem
        U_S1, P_S1 = S.step(dt)

        # Transfer structure displacement to fluidmesh
        M.update_structure_displacement(U_S1)

        # Solve mesh equation
        M.step(dt)

        # Transfer mesh displacement to fluid
        F.update_mesh_displacement(U_M1, dt)
```

```python
# Fluid residual contributions
R_F0 = w*inner(EZ_F - Z_F, Dt_U_F - div(Sigma_F))*dx_F
R_F1 = avg(w)*inner(EZ_F('+') - Z_F('+'),
                    jump(Sigma_F, N_F))*dS_F
R_F2 = w*inner(EZ_F - Z_F, dot(Sigma_F, N_F))*ds
R_F3 = w*inner(EY_F - Y_F,
               div(J(U_M)*dot(inv(F(U_M)), U_F)))*dx_F
```

Computational hemodynamics

```
# Define Cauchy stress tensor
def sigma(v,w):
    return 2.0*mu*0.5*(grad(v) + grad(v).T)  -
w*Identity(v.cell().d)

# Define symmetric gradient
def epsilon(v):
    return  0.5*(grad(v) + grad(v).T)

# Tentative velocity step (sigma formulation)
U = 0.5*(u0 + u)
F1 = rho*(1/k)*inner(v, u - u0)*dx +
rho*inner(v, grad(u0)*(u0 - w))*dx \
   + inner(epsilon(v), sigma(U, p0))*dx \
   + inner(v, p0*n)*ds - mu*inner(grad(U).T*n, v)*ds \
   - inner(v, f)*dx
a1 = lhs(F1)
L1 = rhs(F1)

# Pressure correction
a2 = inner(grad(q), k*grad(p))*dx
L2 = inner(grad(q), k*grad(p0))*dx - q*div(u1)*dx

# Velocity correction
a3 = inner(v, u)*dx
L3 = inner(v, u1)*dx + inner(v, k*grad(p0 - p1))*dx
```

- The Navier–Stokes solver is implemented in Python/FEniCS
- FEniCS allows solvers to be implemented in a minimal amount of code

Valen-Sendstad, Mardal, Logg, *Computational hemodynamics* (2011)

FEniCS under the hood

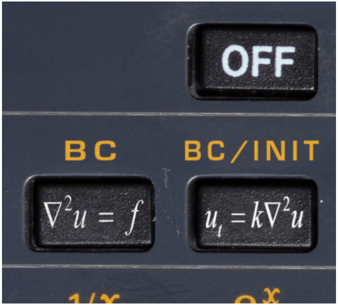
Automated scientific computing

Input

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

Output

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



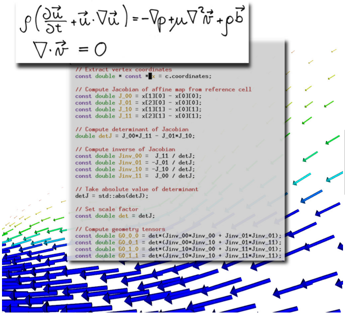
Automatic code generation

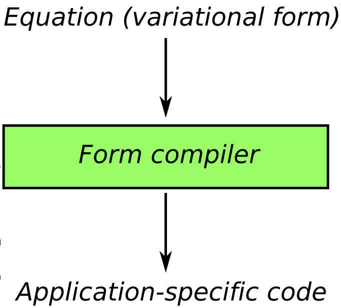
Input

Equation (variational problem)

Output

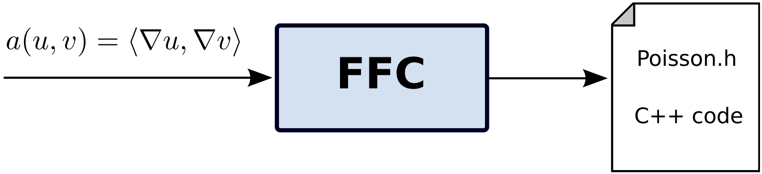
Efficient application-specific code





A common framework: UFL/UFC

- UFL - Unified Form Language
- UFC - Unified Form-assembly Code
- Unify, standardize, extend
- Form compilers: FFC, SyFi



Form compiler interfaces

Command-line

```
>> ffc poisson.ufl
```

Just-in-time

```
V = FunctionSpace(mesh, "CG", 3)
u = TrialFunction(V)
v = TestFunction(V)
A = assemble(dot(grad(u), grad(v))*dx)
```

Code generation system

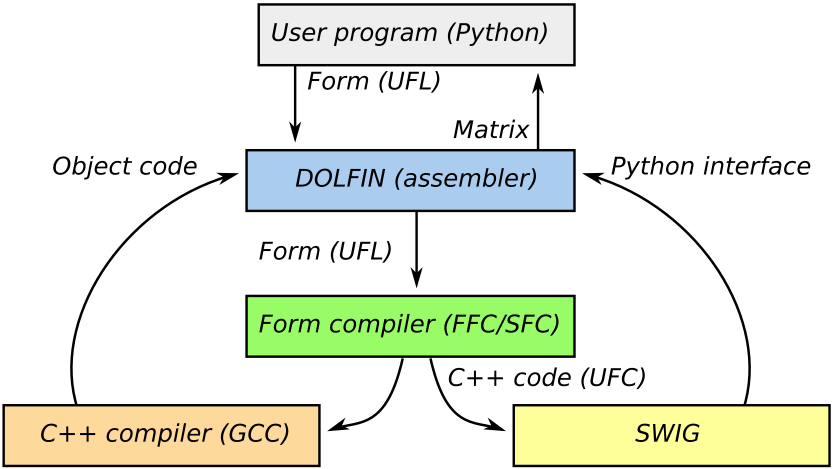
```
mesh = UnitSquare(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]*x[1]")

a = dot(grad(u), grad(v))*dx
L = f*v*dx

bc = DirichletBC(V, 0.0, DomainBoundary())

A = assemble(a)
b = assemble(L)
bc.apply(A, b)

u = Function(V)
solve(A, u.vector(), b)
```

Code generation system

```
mesh = UnitSquare(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]*x[1]")

a = dot(grad(u), grad(v))*dx
L = f*v*dx

bc = DirichletBC(V, 0.0, DomainBoundary())

A = assemble(a)
b = assemble(L)
bc.apply(A, b)

u = Function(V)
solve(A, u.vector(), b)
```

(Python, C++–SWIG–Python, Python–JIT–C++–GCC–SWIG–Python)

Just-In-Time (JIT) compilation

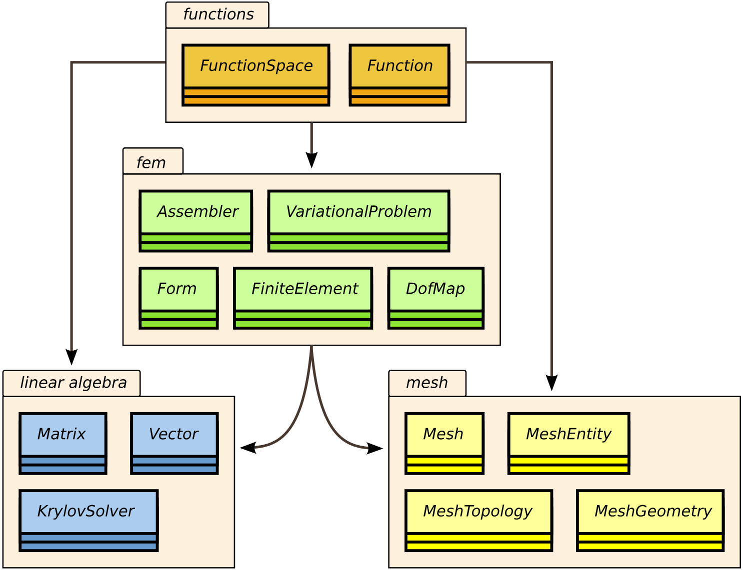


Basic API

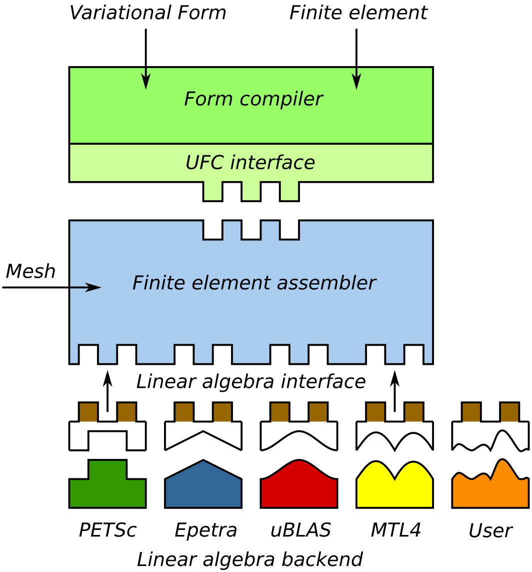
- `Mesh`, `Vertex`, `Edge`, `Face`, `Facet`, `Cell`
- `FiniteElement`, `FunctionSpace`
- `TrialFunction`, `TestFunction`, `Function`
- `grad()`, `curl()`, `div()`, ...
- `Matrix`, `Vector`, `KrylovSolver`, `LUSolver`
- `assemble()`, `solve()`, `plot()`

- Python interface generated semi-automatically by SWIG
- C++ and Python interfaces almost identical

DOLFIN class diagram



Assembler interfaces



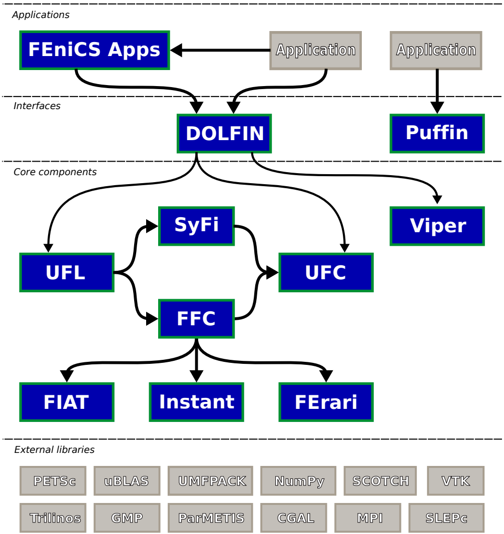
Linear algebra in DOLFIN

- Generic linear algebra interface to
  - PETSc
  - Trilinos/Epetra
  - uBLAS
  - MTL4
- Eigenvalue problems solved by SLEPc for PETSc matrix types
- Matrix-free solvers ("virtual matrices")

Linear algebra backends

```
>>> from dolfin import *
>>> parameters["linear_algebra_backend"] = "PETSc"
>>> A = Matrix()
>>> parameters["linear_algebra_backend"] = "Epetra"
>>> B = Matrix()
```

FEniCS software components



Quality assurance by continuous testing

fenics-buildbot	lucid-amd64	maverick-i386	mac-osx	linux64-exp
	9 (9) / 9	9 (9) / 9	9 (9) / 9	9 (9) / 9
ferari	Success	Success	Success	Success
fiat	Success	Success	Success	Success
ufc	Success	Success	Success	Success
instant	Success	Success	Success	Success
ufl	Success	Success	Success	Success
ffc	Success	Success	Success	building
viper	Success	Success	Success	Success
dolfin	Success	Success	Success	Success
syfi	Success	Success	Success	Success
	9 (9) / 9	9 (9) / 9	9 (9) / 9	9 (9) / 9

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

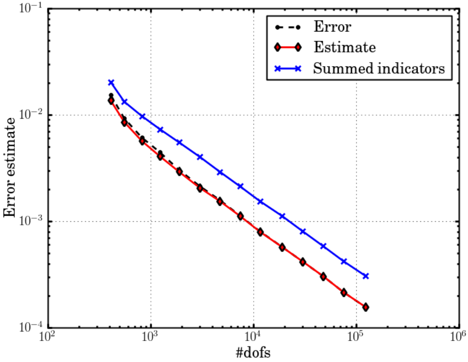
Find $V_h \subset V$ such that $|\mathcal{M}(u) - \mathcal{M}(u_h)| < \epsilon$ where

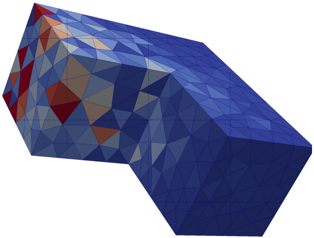
$$a(u_h, v) = L(v) \quad \forall\, v \in V_h$$

Automated in FEniCS (for linear and nonlinear PDE)

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

Poisson's equation

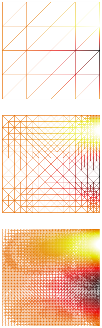


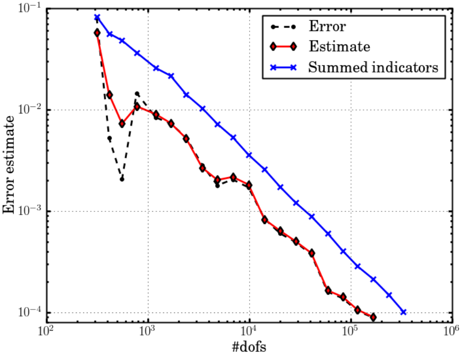


$$a(u, v) = \langle \nabla u, \nabla v \rangle$$

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

A three-field mixed elasticity formulation

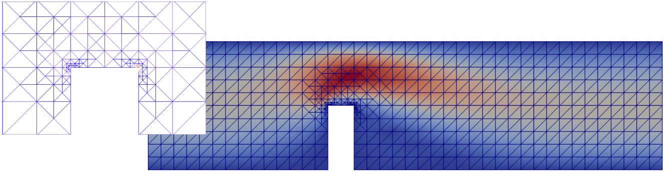




$$a((\sigma, u, \gamma), (\tau, v, \eta)) = \langle A\sigma, \tau\rangle + \langle u, \operatorname{div} \tau\rangle + \langle \operatorname{div} \sigma, v\rangle + \langle \gamma, \tau\rangle + \langle \sigma, \eta\rangle$$

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

Incompressibe Navier–Stokes



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

Uniform

1.000.000 dofs, N hours

Adaptive

5.200 dofs, 127 seconds

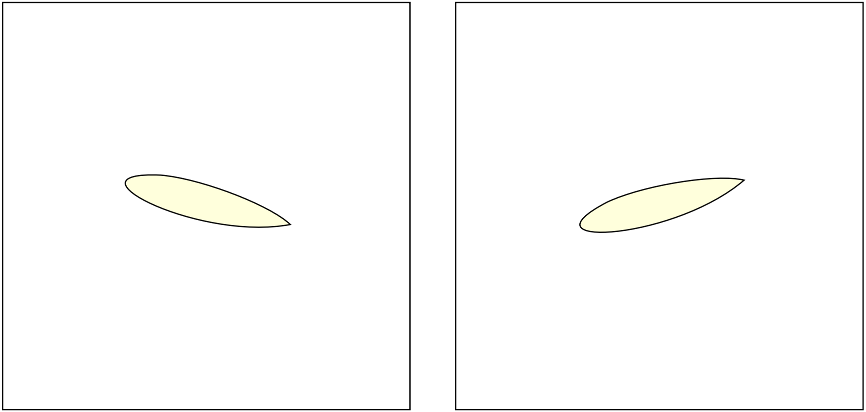
```
from dolfin import *

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* (2010)

Overlapping non-matching meshes

Simulation on overlapping non-matching meshes

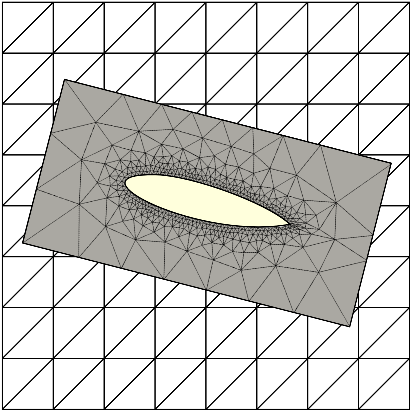
,



Massing, Larson, Logg, Rognes, *A Nitsche overlapping mesh method for the Stokes problem* (2012)

Simulation on overlapping non-matching meshes

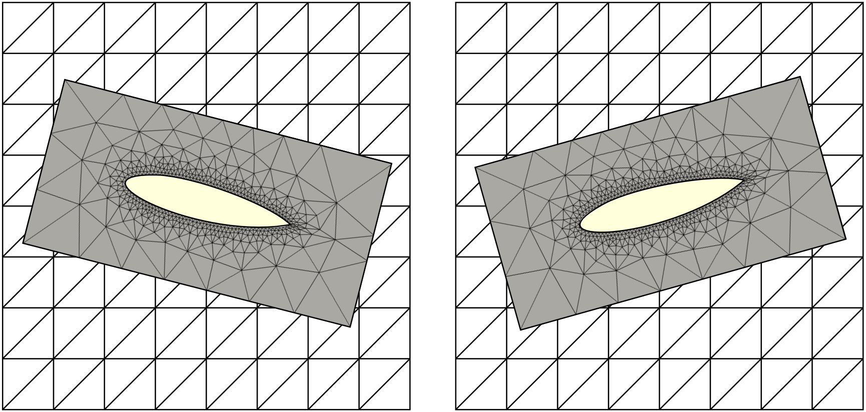
,



Massing, Larson, Logg, Rognes, *A Nitsche overlapping mesh method for the Stokes problem* (2012)

Simulation on overlapping non-matching meshes

,



Massing, Larson, Logg, Rognes, *A Nitsche overlapping mesh method for the Stokes problem* (2012)

A Nitsche formulation for the Stokes problem

Variational formulation

Find $(\boldsymbol{u}_h, p_h) \in V_h^k \times Q_h^l$ such that $\forall\, (\boldsymbol{v}_h, q_h) \in V_h^k \times Q_h^l$:

$$a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) + b_h(\boldsymbol{v}_h, p_h) + b_h(\boldsymbol{u}_h, q_h) + s_h(\boldsymbol{u}_h, \boldsymbol{v}_h) - S_h(\boldsymbol{u}_h, p_h; \boldsymbol{v}_h, q_h) = L_h(\boldsymbol{v}_h, q_h)$$

where

$$a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) = (\nabla \boldsymbol{u}_h, \nabla \boldsymbol{v}_h)_{\Omega_1 \cup \Omega_2} \underbrace{- (\langle \partial_{\boldsymbol{n}} \boldsymbol{u}_h \rangle, [\boldsymbol{v}_h])_\Gamma}_{\text{Nitsche terms}}$$

$$\underbrace{- (\langle \partial_{\boldsymbol{n}} \boldsymbol{v}_h \rangle, [\boldsymbol{u}_h])_\Gamma + \gamma (h^{-1} [\boldsymbol{u}_h], [\boldsymbol{v}_h])_\Gamma}_{\text{Nitsche terms}},$$

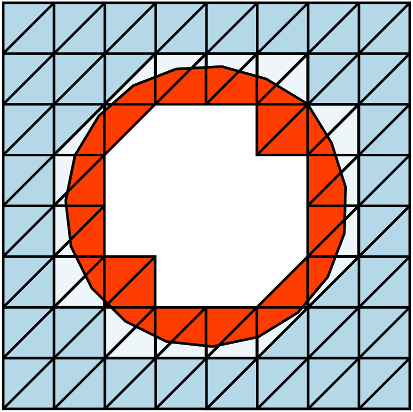
$$b_h(\boldsymbol{v}_h, q_h) = -(\nabla \cdot \boldsymbol{v}_h, q_h)_{\Omega_1 \cup \Omega_2} + \underbrace{(\boldsymbol{n} \cdot [\boldsymbol{v}_h], \langle q_h \rangle)_\Gamma}_{\text{Nitsche terms}},$$

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

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

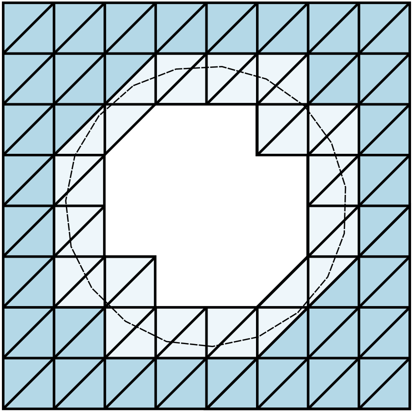
$$L_h(\boldsymbol{v}, q) = (\boldsymbol{f}, \boldsymbol{v}) - \delta \sum_{T \in \mathcal{T}_1^* \cup \mathcal{T}_2} h_T^2 (\boldsymbol{f}, -\alpha \Delta \boldsymbol{v}_h + \beta \nabla q_h)_T.$$

Ghost-penalties added in the interface zone



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

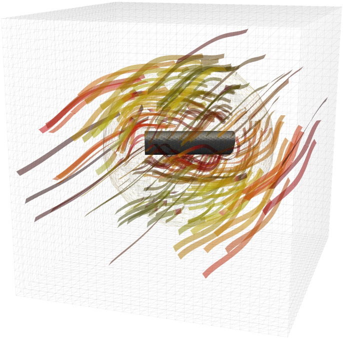
Ghost penalty for u



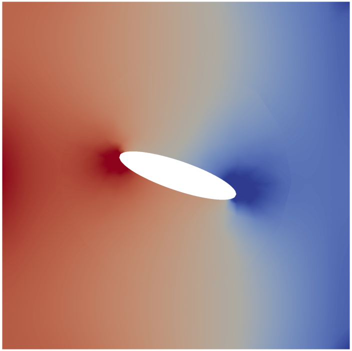
$$\delta \sum_{T \in \mathcal{T}_1^* \cup \mathcal{T}_2} h_T^2(-\Delta \boldsymbol{u}_h + \nabla p_h, -\alpha \Delta \boldsymbol{v}_h + \beta \nabla q_h)_T$$

Ghost penalty for p

Stokes flow for different angles of attack

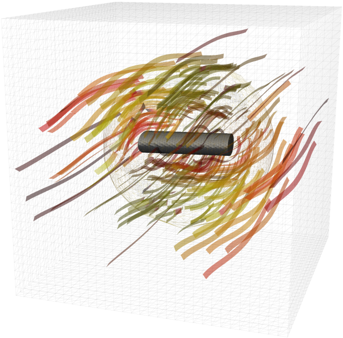


Velocity streamlines

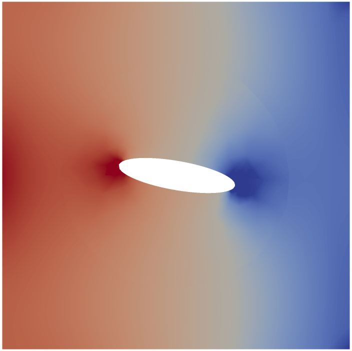


Pressure

Stokes flow for different angles of attack

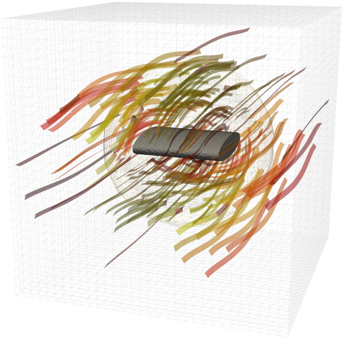


Velocity streamlines

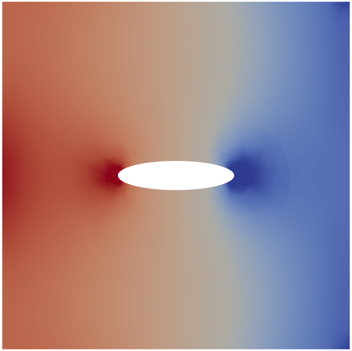


Pressure

Stokes flow for different angles of attack

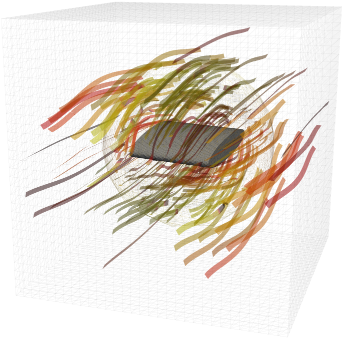


Velocity streamlines

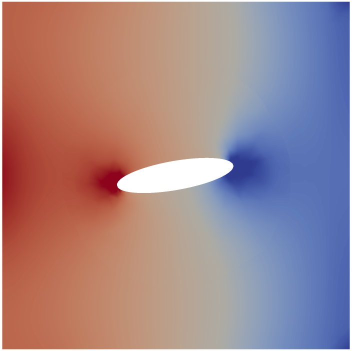


Pressure

Stokes flow for different angles of attack

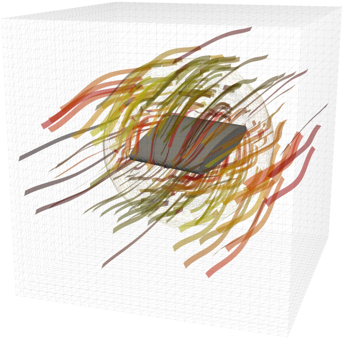


Velocity streamlines

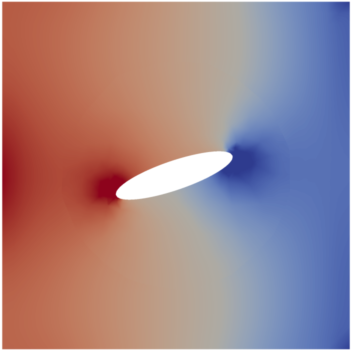


Pressure

Stokes flow for different angles of attack



Velocity streamlines



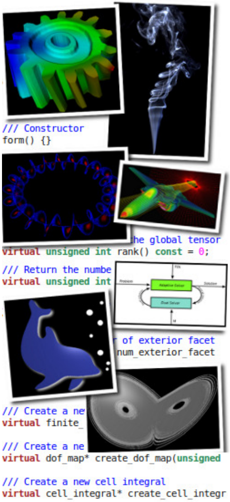
Pressure

Closing remarks

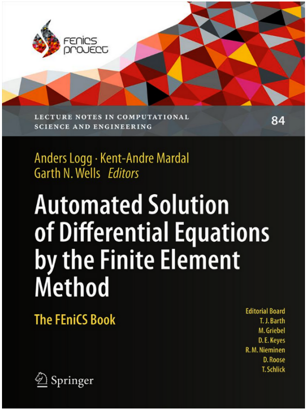
Summary

- Automated solution of PDE
- Easy install
- Easy scripting in Python
- Efficiency by automated code generation
- Free/open-source (LGPL)

`http://fenicsproject.org/`



Current and future plans



- Parallelization (2009)
- Automated error control (2010)
- Debian/Ubuntu (2010)
- Documentation (2011)
- FEniCS 1.0 (2011)
- The FEniCS Book (2012)

- FEniCS'12 at Simula (June 2012)
- Visualization, mesh generation
- Parallel AMR
- Hybrid MPI/OpenMP
- Overlapping/intersecting meshes