FEniCS Course

Lecture 1: Introduction

Contributors
Anders Logg

Course outline

L1 Introduction

L2 Static linear PDEs

L3 Static nonlinear PDEs

L4 Time-dependent PDEs

L5 Advanced topics

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

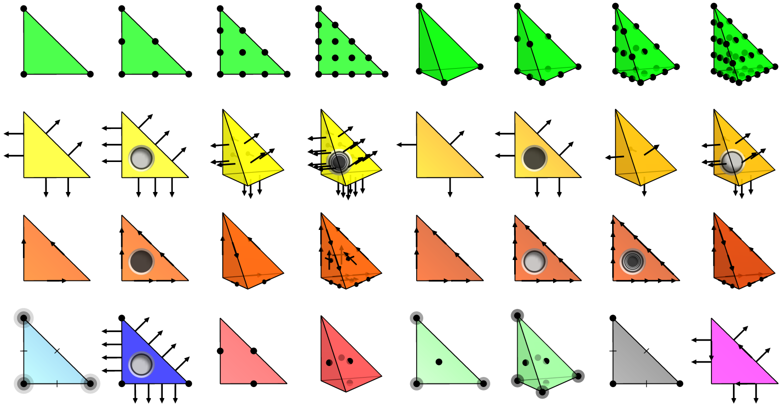
`http://fenicsproject.org/`

Collaborators

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

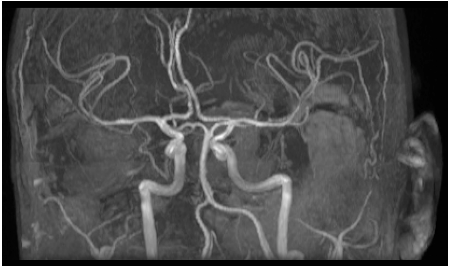
FEniCS is automated FEM

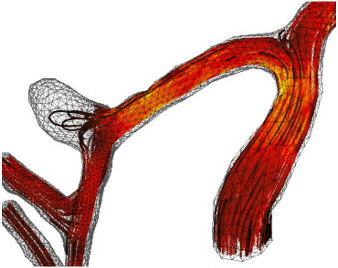
- Automated generation of basis functions
- Automated evaluation of variational forms
- Automated finite element assembly
- Automated adaptive error control



What has FEniCS been used for?

Computational hemodynamics





- Low wall shear stress may trigger aneurysm growth
- Solve the incompressible Navier–Stokes equations on patient-specific geometries

$$\begin{aligned} \dot{u} + u \cdot \nabla u - \nabla \cdot \sigma(u, p) &= f \\ \nabla \cdot u &= 0 \end{aligned}$$

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

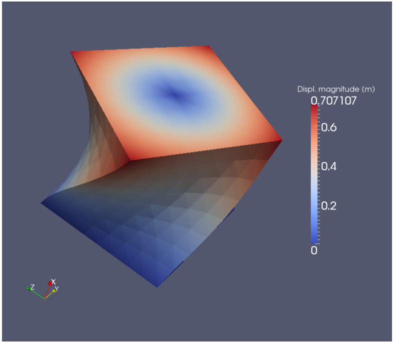
Computational hemodynamics (contd.)

```
# 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)

Hyperelasticity




```
class Twist(StaticHyperelasticity):

    def mesh(self):
        n = 8
        return UnitCube(n, n, n)

    def dirichlet_conditions(self):
        clamp = Expression(("0.0", "0.0",
            "0.0"))
        twist = Expression(("0.0",
          "y0 + (x[1]-y0)*cos(theta)
              - (x[2]-z0)*sin(theta) - x[1]",
          "z0 + (x[1]-y0)*sin(theta)
              + (x[2]-z0)*cos(theta) - x[2]"))
        twist.y0 = 0.5
        twist.z0 = 0.5
        twist.theta = pi/3
        return [clamp, twist]

    def dirichlet_boundaries(self):
        return ["x[0] == 0.0", "x[0] == 1.0"]

    def material_model(self):
        mu    = 3.8461
        lmbda =
            Expression("x[0]*5.8+(1-x[0])*5.7")

        material = StVenantKirchhoff([mu,
            lmbda])
        return material

    def __str__(self):
        return "A cube twisted by 60 degrees"
```

- CBC.Solve is a collection of FEniCS-based solvers developed at CBC
- CBC.Twist, CBC.Flow, CBC.Swing, CBC.Beat, ...

H. Narayanan, *A computational framework for nonlinear elasticity* (2011)

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)
```

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

Course outline

L1 Introduction

L2 Static linear PDEs

L3 Static nonlinear PDEs

L4 Time-dependent PDEs

L5 Advanced topics