Geilo Winter School 2012

Lecture 4: Introduction to FEniCS

Anders Logg

Course outline

Sunday

L1 Introduction to FEM

Monday

L2 Fundamentals of continuum mechanics (I)

L3 Fundamentals of continuum mechanics (II)

L4 Introduction to FEniCS

Tuesday

L5 Solid mechanics

L5 Static hyperelasticity in FEniCS

L5 Dynamic hyperelasticity in FEniCS

Wednesday

L5 Fluid mechanics

L5 Navier–Stokes in FEniCS

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

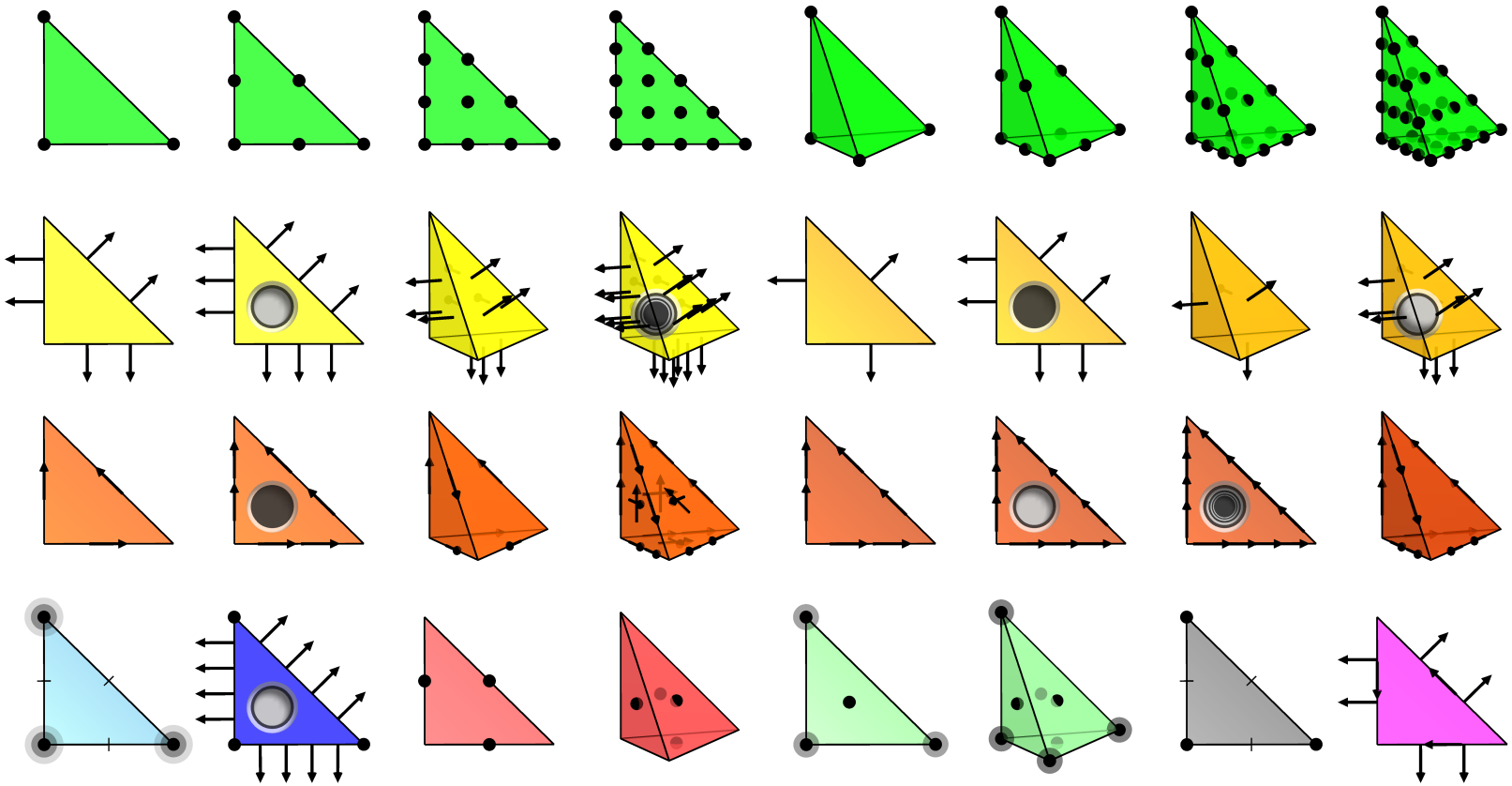
`http://fenicsproject.org/`

Collaborators

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

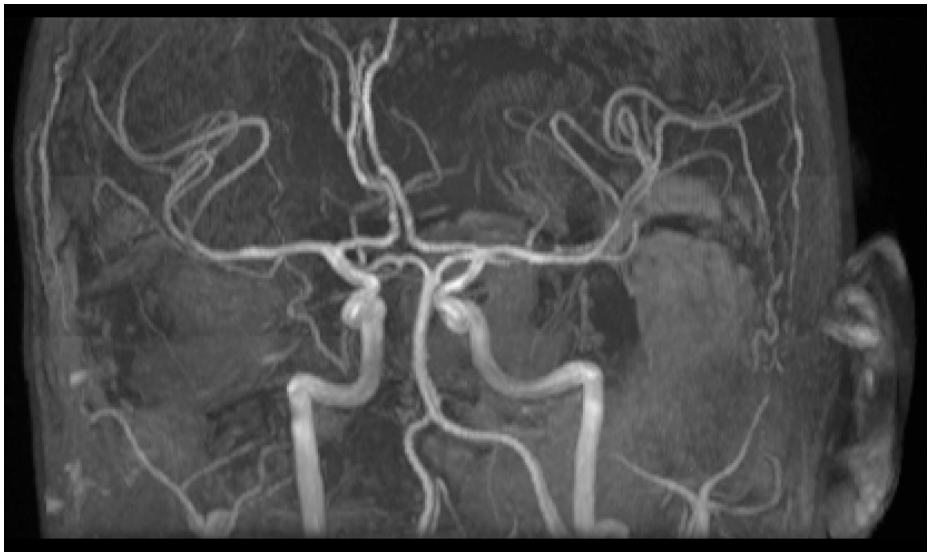
FEniCS is automated FEM

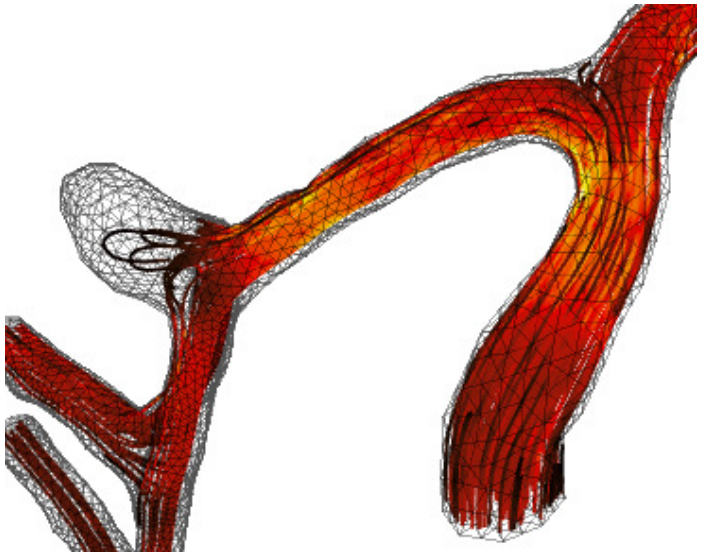
- Automated generation of basis functions
- Automated evaluation of variational forms
- Automated finite element assembly
- Automated adaptive error control



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

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

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

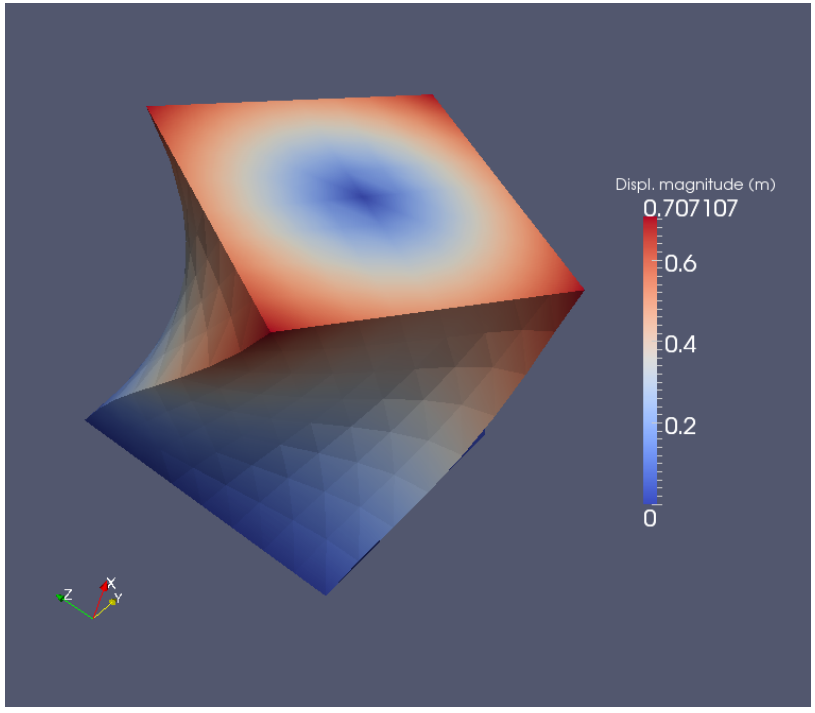
Computational hemodynamics (contd.)

```
# 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

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

Hello World!

We will solve Poisson's equation, the Hello World of scientific computing:

$$\begin{aligned} -\Delta u &= f \quad && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}$$

Poisson's equation arises in numerous applications:

- heat conduction, electrostatics, diffusion of substances, twisting of elastic rods, inviscid fluid flow, water waves, magnetostatics
- as part of numerical splitting strategies for more complicated systems of PDEs, in particular the Navier–Stokes equations

Solving PDEs in FEniCS

Solving a physical problem with FEniCS consists of the following steps:

1. Identify the PDE and its boundary conditions
2. Reformulate the PDE problem as a variational problem
3. Make a Python program where the formulas in the variational problem are coded, along with definitions of input data such as f, u_0, and a mesh for Ω
4. Add statements in the program for solving the variational problem, computing derived quantities such as ∇u, and visualizing the results

Deriving a variational problem for Poisson's equation

The simple recipe is: multiply the PDE by a test function v and integrate over Ω:

$$-\int_\Omega (\Delta u) v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

Then integrate by parts and set $v = 0$ on the Dirichlet boundary:

$$-\int_\Omega (\Delta u) v \,\mathrm{d}x = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x - \underbrace{\int_{\partial\Omega} \frac{\partial u}{\partial n} v \,\mathrm{d}s}_{=0}$$

We find that:

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

Variational problem for Poisson's equation

Find $u \in V$ such that

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

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

The trial space V and the test space $\hat{V}$ are (here) given by

$$V = \{v \in H^1(\Omega) : v = u_0 \text{ on } \partial\Omega\}$$
$$\hat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$$

Discrete variational problem for Poisson's equation

We approximate the continuous variational problem with a discrete variational problem posed on finite dimensional subspaces of V and $\hat{V}$:

$$V_h \subset V$$
$$\hat{V}_h \subset \hat{V}$$

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

$$\int_\Omega \nabla u_h \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$$

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

Canonical variational problem

The following canonical notation is used in FEniCS: find $u \in V$ such that

$$a(u, v) = L(v)$$

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

For Poisson's equation, we have

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

$a(u, v)$ is a *bilinear form* and $L(v)$ is a *linear form*

A test problem

We construct a test problem for which we can easily check the answer. We first define the exact solution by

$$u(x, y) = 1 + x^2 + 2y^2$$

We insert this into Poisson's equation:

$$f = -\Delta u = -\Delta(1 + x^2 + 2y^2) = -(2 + 4) = -6$$

This technique is called the *method of manufactured solutions*

Implementation in FEniCS

```
from dolfin import *

mesh = UnitSquare(6, 4)
V = FunctionSpace(mesh, "Lagrange", 1)
u0 = Expression("1 + x[0]*x[0] + 2*x[1]*x[1]")

def u0_boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u0, u0_boundary)

f = Constant(-6.0)
u = TrialFunction(V)
v = TestFunction(V)
a = inner(grad(u), grad(v))*dx
L = f*v*dx

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

Step by step: the first line

The first line of a FEniCS program usually begins with

```
from dolfin import *
```

This imports key classes like `UnitSquare`, `FunctionSpace`, `Function` and so forth, from the FEniCS user interface (DOLFIN)

Step by step: creating a mesh

Next, we create a mesh of our domain Ω:

```
mesh = UnitSquare(6, 4)
```

This defines a mesh of $6 \times 4 \times 2 = 48$ triangles of the unit square

Other useful classes for creating meshes include `UnitInterval`, `UnitCube`, `UnitCircle`, `UnitSphere`, `Rectangle` and `Box`

Step by step: creating a function space

The following line creates a function space on Ω:

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

The second argument reflects the type of element, while the third argument is the degree of the basis functions on the element

Other types of elements include `"Discontinuous Lagrange"`, `"Brezzi-Douglas-Marini"`, `"Raviart-Thomas"`, `"Crouzeix-Raviart"`, `"Nedelec 1st kind H(curl)"` and `"Nedelec 2nd kind H(curl)"`

Step by step: defining expressions

Next, we define an expression for the boundary value:

```
u0 = Expression("1 + x[0]*x[0] + 2*x[1]*x[1]")
```

The formula must be written in C++ syntax

The `Expression` class is very flexible and can be used to create complex user-defined expressions. For more information, try

```
help(Expression)
```

in Python or, in the shell:

```
pydoc dolfin.Expression
```

Step by step: defining boundaries

We next define the Dirichlet boundary:

```
def u0_boundary(x, on_boundary):
    return on_boundary
```

You may want to experiment with the definition of the boundary:

```
def u0_boundary(x):
    return x[0] < DOLFIN_EPS or \
           x[1] > 1.0 - DOLFIN_EPS
```

```
def u0_boundary(x):
    return near(x[0], 0.0) or near(x[1], 1.0)
```

```
def u0_boundary(x, on_boundary):
    return on_boundary and x[0] > DOLFIN_EPS
```

Step by step: defining a boundary condition

The following code defines a Dirichlet boundary condition:

```
bc = DirichletBC(V, u0, u0_boundary)
```

This boundary condition states that a function in the function space defined by `V` should be equal to `u0` on the boundary defined by `u0_boundary`

Note that the above line does not yet apply the boundary condition to all functions in the function space

Step by step: defining the right-hand side

The right-hand side $f = 6$ may be defined as follows:

```
f = Expression("-6")
```

or (more efficiently) as

```
f = Constant(-6.0)
```

Step by step: defining variational problems

Variational problems are defined in terms of *trial* and *test* functions:

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

We now have all the objects we need in order to specify the bilinear form $a(u, v)$ and the linear form $L(v)$:

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

Step by step: solving variational problems

Once a variational problem has been defined, it may be solved by calling the `solve` function:

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

Note the reuse of the variable `u` as both a `TrialFunction` in the variational problem and a `Function` to store the solution

Step by step: post-processing

The solution and the mesh may be plotted by simply calling:

```
plot(u)
plot(mesh)
interactive()
```

The `interactive()` call is necessary for the plot to remain on the screen and allows the plots to be rotated, translated and zoomed

For postprocessing in ParaView or MayaVi, store the solution in VTK format:

```
file = File("poisson.pvd")
file << u
```

The FEniCS challenge!

Solve the partial differential equation

$$-\Delta u = f$$

with homogeneous Dirichlet boundary conditions on the unit square for $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$. Plot the error in the L^2 norm as function of the mesh size h for a sequence of refined meshes. Try to determine the convergence rate.

- *Who can obtain the smallest error?*
- *Who can compute a solution with an error smaller than* $\epsilon = 10^{-6}$ *in the fastest time?*

The best students(s) will be rewarded with an exclusive FEniCS coffee mug!