FEniCS Course

Overview and Introduction

Lecturer
Simon Funke
Berlin, January 23–24 2018

Course outline

Tue ⋆ Overview and Introduction

L01 Installation of FEniCS

L02 Static linear PDEs

L04 Time-dependent PDEs

Wed ⋆ PDE-constrained optimisation

L13 Introduction to dolfin-adjoint

L14 From sensitivity to optimisation

L16 Optimisation challenge

The lectures can be downloaded from http://simonfunke.com/fenics-lecture

Full list of FEniCS lectures

All lectures can be downloaded from

`http://fenicsproject.org/pub/course/`

The FEniCS Project is a collection of open-source software components aimed at the numerical solution of partial differential equations using finite element methods

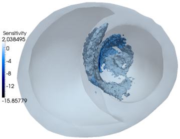
Key distinguishing features

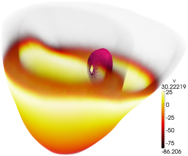
- FEniCS (Python/C++) code is quick to write and easy to read
- ‘Any’ finite element formulation of ’any’ partial differential equation can be coded
- Automated code generation is heavily used under the hood to create efficient, specialized, low-level code
- Performance – implicit problems with over 12 000 000 000 degrees of freedom can be solved in a couple of minutes

http://fenicsproject.org/

FEniCS has been used for a wide range of equations and applications

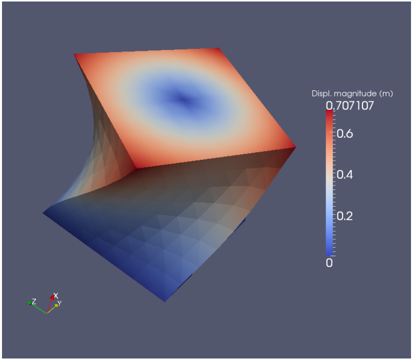
Reaction-diffusion equations; Stokes with or without nonlinear viscosity; compressible and incompressible Navier–Stokes; RANS turbulence models; shallow water equations; Bidomain equations; nonlinear and linear elasticity; nonlinear and linear viscoelasticity; Schrödinger; Biot's equations for porous media, fracture mechanics, electromagnetism, liquid crystals including liquid crystal elastomers, combustion, ... and coupled systems of the above, ...





for simulating blood flow, computing calcium release in cardic tissue, computing the cardiac potential in the heart, simulating mantle convection, simulating melting ice sheets, computing the optimal placement of tidal turbines, simulating and reconstructing tsunamis, simulating the flow of cerebrospinal fluid and the deformation of the spinal cord, simulating waveguides, ...

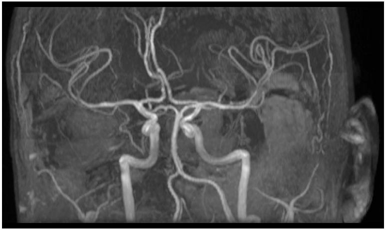
Hyperelasticity

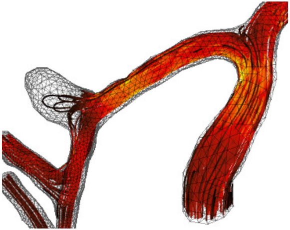



```
from fenics import *

mesh = UnitCubeMesh(24, 16, 16)
V = VectorFunctionSpace(mesh, "Lagrange", 1)

left =  CompiledSubDomain("(std::abs(x[0])
    < DOLFIN_EPS) && on_boundary")
right = CompiledSubDomain("(std::abs(x[0] - 1.0)
    < DOLFIN_EPS) && on_boundary")

c = Expression(("0.0", "0.0", "0.0"), degree=0)
r = Expression(("0.0",
"0.5*(y0+(x[1]-y0)*cos(t)-(x[2]-z0)*sin(t)-x[1])",
"0.5*(z0+(x[1]-y0)*sin(t)+(x[2]-z0)*cos(t)-x[2])"),
y0=0.5, z0=0.5, t=pi/3, degree=3)
bcl = DirichletBC(V, c, left)
bcr = DirichletBC(V, r, right)
bcs = [bcl, bcr]
v  = TestFunction(V)
u  = Function(V)
B  = Constant((0.0, -0.5, 0.0))
T  = Constant((0.1,  0.0, 0.0))
I = Identity(V.cell().d)
F = I + grad(u)
Ic = tr(F.T*F)
J  = det(F)
E, nu = 10.0, 0.3
mu, lmbda = Constant(E/(2*(1 + nu))),
    Constant(E*nu/((1 + nu)*(1 - 2*nu)))
psi = (mu/2)*(Ic - 3) - mu*ln(J) +
    (lmbda/2)*(ln(J))**2
Pi = psi*dx - dot(B, u)*dx - dot(T, u)*ds
F = derivative(Pi, u, v)

solve(F == 0, u, bcs)
plot(u, interactive=True, mode="displacement")
```

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

Computational hemodynamics





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

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

- Use PDE-constrained optimisation to assimilate measurements into simulation.

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

Funke, Nordaas, Evju, Alnæs, Mardal, *arxiv* (2018)

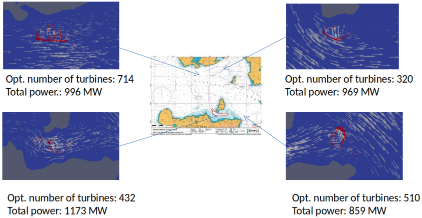
Computational hemodynamics (contd.)

```python
# 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)

Simulation and optimisation of tidal turbine arrays



- Tidal turbine arrangement influences the total performance of an array.
- Solve the shallow water equations, and compute optimal design.

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

Funke, Kramer, Farrell, Piggott, *Renewable Energy* (2014, 2016)

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

```python
from fenics import *

mesh = UnitSquareMesh(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]*x[1]", degree=2)

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

What happens behind the scences?

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

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

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

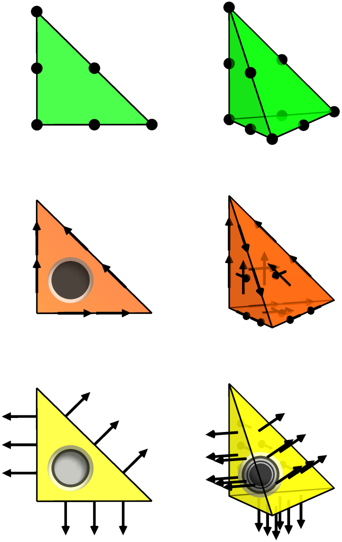
$$\int_\Omega 2\nu(u,T)\varepsilon(u)\cdot\varepsilon(v) + \operatorname{div}(v)\,p + \operatorname{div}(u)\,q - \mathrm{Ra} T g \cdot v \,\mathrm{d}x = 0$$

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

```python
from fenics 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

```python
from fenics 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)
```

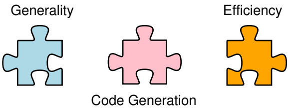


Kirby 2004; Ølgaard, Logg, and Wells 2008; R., Kirby and Logg 2009

FEniCS provides an expressive form language close to mathematical syntax

```
from fenics 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

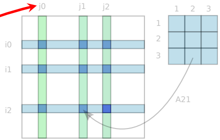


Kirby and Logg 2006; Alnæs 2012

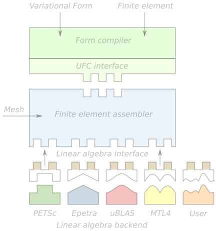
FEniCS provides automated form assembly over finite element meshes and numerical linear algebra

```python
from fenics 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

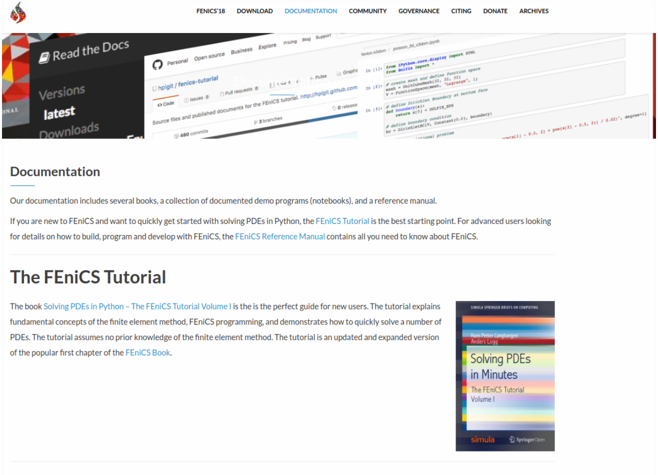
Sounds great, but how do I find my way through the jungle?

Three survival advices

Use the right Python tools

Explore the documentation

Ask, report and request


FENICS'18 DOWNLOAD DOCUMENTATION COMMUNITY GOVERNANCE CITING DONATE ARCHIVES

Documentation

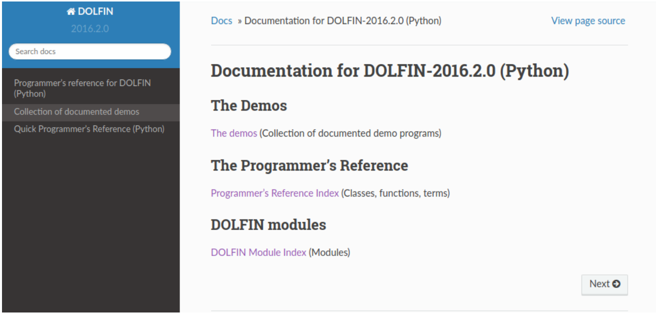
Our documentation includes several books, a collection of documented demo programs (notebooks), and a reference manual.

If you are new to FEniCS and want to quickly get started with solving PDEs in Python, the FEniCS Tutorial is the best starting point. For advanced users looking for details on how to build, program and develop with FEniCS, the FEniCS Reference Manual contains all you need to know about FEniCS.

The FEniCS Tutorial

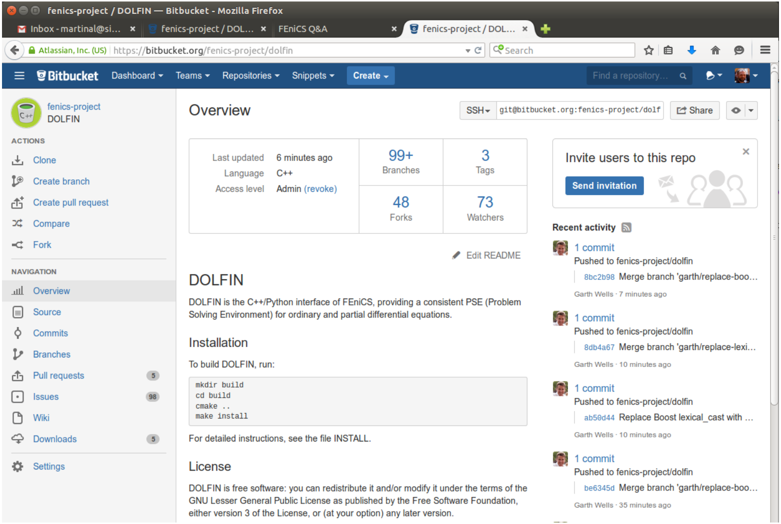
The book Solving PDEs in Python – The FEniCS Tutorial Volume I is the is the perfect guide for new users. The tutorial explains fundamental concepts of the finite element method, FEniCS programming, and demonstrates how to quickly solve a number of PDEs. The tutorial assumes no prior knowledge of the finite element method. The tutorial is an updated and expanded version of the popular first chapter of the FEniCS Book.

http://fenicsproject.org/documentation
https://fenicsproject.org/tutorial



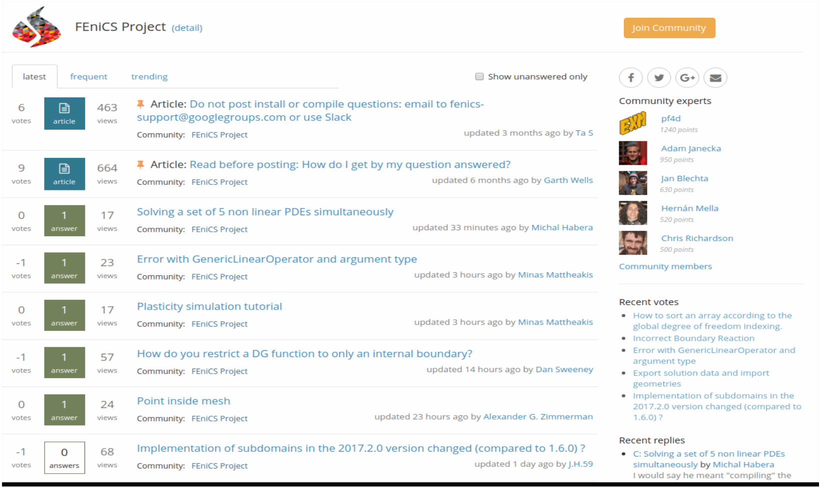
https://fenicsproject.org/olddocs/dolfin/2016.2.0/python/

Development community is organized via bitbucket.org



http://bitbucket.org/fenics-project/

Community help is available via QA forum



`allanswered.com/community/s/fenics-project` (new)

`fenicsproject.org/qa` (old)

Let's get started and remember:

- **Lectures** can be downloaded from

  `simonfunke.com/p/fenics-course-berlin.html`

- **Data** for exercises can be downloaded from

  `http://fenicsproject.org/pub/course/data`

- **Solutions** for exercises can be downloaded from

  `https://github.com/funsim/hub_fenics_workshop/tree/master/solutions`

Installation alternatives

☞ Docker images on Linux, Mac, Windows

☞ PPA with apt packages for Debian and Ubuntu

☞ Anaconda packages

☞ Build from source

http://fenicsproject.org/download/

The FEniCS challenge!

1. Install FEniCS on your laptop!

   `http://fenicsproject.org/download/`

2. Download and execute demo_cahn-hilliard.py, try to visualize the results with Paraview.
3. Which elements are supported in `dolfin`
   Hint: Check documentation of *dolfin.FunctionSpace*?

See also **`https://femtable.org`**