# The FEniCS Project

Anders Logg

Simula Research Laboratory / University of Oslo

Det Norske Veritas

2011–05–11

# The FEniCS Project

*Free Software for Automated Scientific Computing*

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

`http://www.fenicsproject.org/`

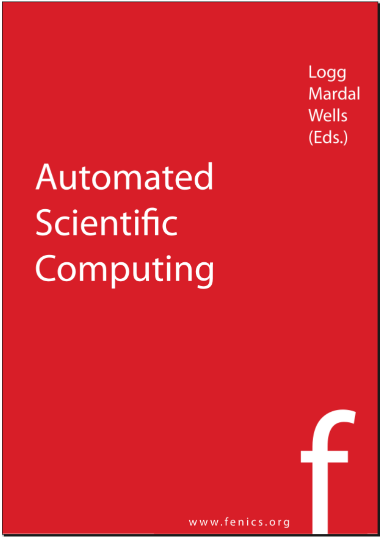
**Collaborators**

*University of Chicago, Argonne National Laboratory, Delft University of Technology, Royal Institute of Technology KTH, Simula Research Laboratory, Texas Tech University, University of Cambridge, . . .*

# Key Features

- Simple and intuitive object-oriented API, C++ or Python
- Automatic and efficient evaluation of variational forms
- Automatic and efficient assembly of linear systems
- Distributed (clusters) and shared memory (multicore) parallelism
- General families of finite elements, including arbitrary order continuous and discontinuous Lagrange elements, BDM, RT, Nedelec, ...
- Arbitrary mixed elements
- High-performance parallel linear algebra
- General meshes, adaptive mesh refinement
- mcG($q$)/mdG($q$) and cG($q$)/dG($q$) ODE solvers
- Support for a range of input/output formats
- Built-in plotting

# The State of FEniCS




- Parallelization (2009)
- Automated error control (2010)
- Debian/Ubuntu (2010)
- Documentation (2010)
- Latest release: 0.9.10 (Feb 2011)

- Release of 1.0 (2011)
- Book (2011)
- New web page (2011)

# Outline

- Automated Scientific Computing
- Interface and Design
- Examples and Applications

# Automated Scientific Computing

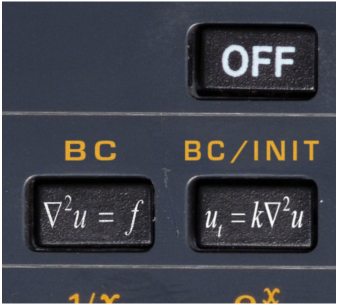
# Automated Scientific Computing

**Input**

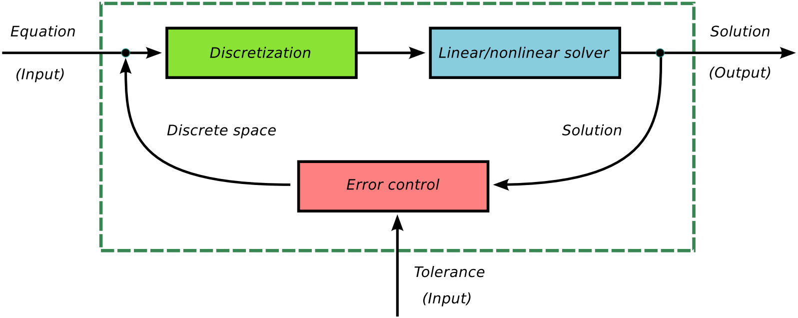
- $A(u) = f$
- $\epsilon > 0$

**Output**

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



# Blueprint


Equation
(Input)
Discretization
Linear/nonlinear solver
Solution
(Output)
Discrete space
Solution
Error control
Tolerance
(Input)

# Key Steps

## Key steps

(i) Automated discretization ✓ (2006)

(ii) Automated error control ✓ (2010)

(iii) Automated discrete solution . . .

## Key techniques

- Adaptive finite element methods
- **Automatic code generation**

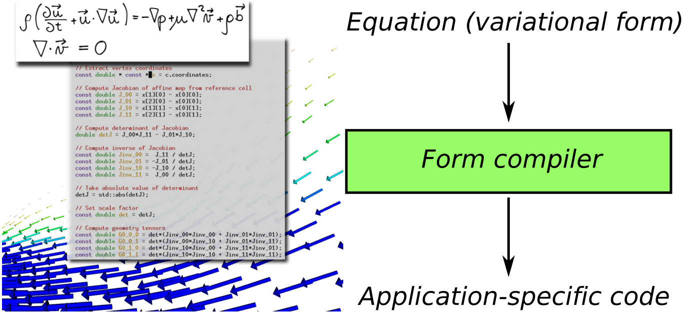
# Automatic Code Generation

**Input**

Equation (variational problem)

**Output**

Efficient application-specific code



# Speedup

- CPU time for computing the "element stiffness matrix"
- Straight-line C++ code generated by the FEniCS Form Compiler (FFC)
- Speedup vs a standard quadrature-based C++ code with loops over quadrature points
- Recently, optimized quadrature code has been shown to be competitive [Oelgaard/Wells, TOMS 2009]

| Form | $q = 1$ | $q = 2$ | $q = 3$ | $q = 4$ | $q = 5$ | $q = 6$ | $q = 7$ | $q = 8$ |
|---|---|---|---|---|---|---|---|---|
| Mass 2D | 12 | 31 | 50 | 78 | 108 | 147 | 183 | 232 |
| Mass 3D | 21 | 81 | 189 | 355 | 616 | 881 | 1442 | 1475 |
| Poisson 2D | 8 | 29 | 56 | 86 | 129 | 144 | 189 | 236 |
| Poisson 3D | 9 | 56 | 143 | 259 | 427 | 341 | 285 | 356 |
| Navier–Stokes 2D | 32 | 33 | 53 | 37 | — | — | — | — |
| Navier–Stokes 3D | 77 | 100 | 61 | 42 | — | — | — | — |
| Elasticity 2D | 10 | 43 | 67 | 97 | — | — | — | — |
| Elasticity 3D | 14 | 87 | 103 | 134 | — | — | — | — |

# Interface and Design

# A Simple Example

$$-\Delta u + u = f \quad \text{in } \Omega$$
$$\partial_n u = 0 \quad \text{on } \partial\Omega$$

## Canonical variational problem

Find $u \in V$ such that

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

where

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

Here: $V = \hat{V} = H^1(\Omega)$, $f(x, y) = \sin x \, \sin y$, $\Omega = (0, 1) \times (0, 1)$

# Programming in FEniCS

## Complete code (Python)

```
from dolfin import *

# Define variational problem
mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("sin(x[0])*sin(x[1])")
a = (grad(u), grad(v)) + (u, v)
L = (f, v)

# Compute and plot solution
problem = VariationalProblem(a, L)
u = problem.solve()
plot(u)
```

# Design Considerations

- Simple and minimal interfaces
- Efficient backends

- Object-oriented API (but not too much)
- Code genereration (but not too much)

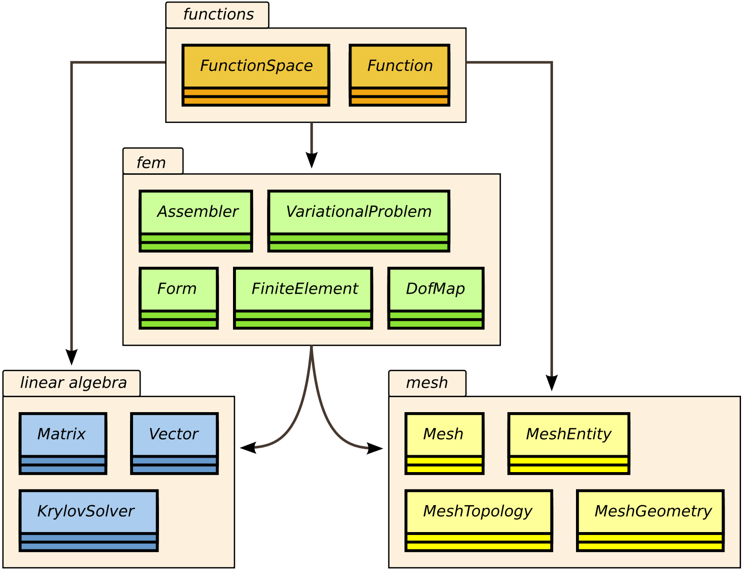
- Application-driven development
- Technology-driven development

# Basic API

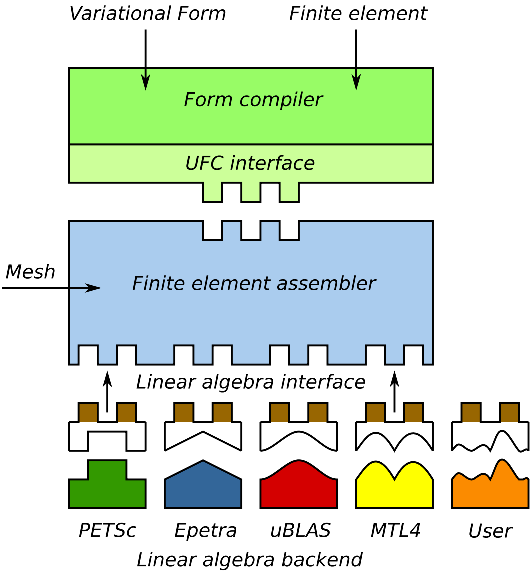
- `Mesh`, `MeshEntity`, `Vertex`, `Edge`, `Face`, `Facet`, `Cell`
- `FiniteElement`, `FunctionSpace`
- `TrialFunction`, `TestFunction`, `Function`
- `grad()`, `curl()`, `div()`, `...`
- `Matrix`, `Vector`, `KrylovSolver`
- `assemble()`, `solve()`, `plot()`

- 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()
```

# Code Generation System

```
from dolfin import *

mesh = UnitSquare(32, 32)

V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("sin(x[0])*sin(x[1])")
a = (grad(u), grad(v)) + (u, v)
L = (f, v)

A = assemble(a, mesh)
b = assemble(L, mesh)

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

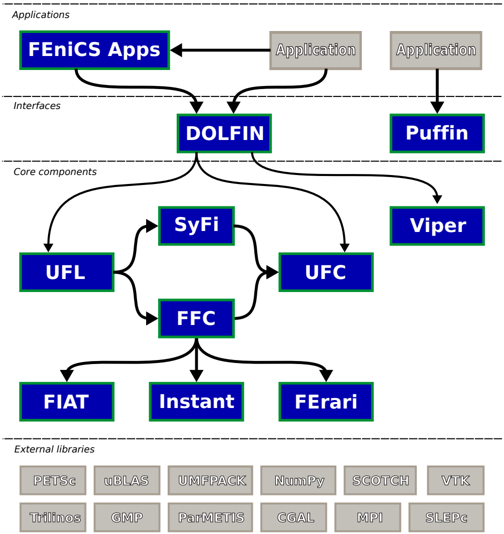
(Python, C++ – SWIG – Python, Python – JIT – C++ – GCC – SWIG – Python)

# Code Generation System

```
from dolfin import *

mesh = UnitSquare(32, 32)

V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("sin(x[0])*sin(x[1])")
a = (grad(u), grad(v)) + (u, v)
L = (f, v)

A = assemble(a, mesh)
b = assemble(L, mesh)

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

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

# FEniCS Software Components



# Installation

Official packages for Debian and Ubuntu

Drag and drop installation on Mac OS X (requires XCode)

Binary installer for Windows (based on MinGW)

- Automated building from source for a multitude of platforms (using Dorsal)
- VirtualBox / VMWare + Ubuntu!

# Nightly Testing

| fenics-buildbot | hardy-i386 | jaunty-amd64 | mac-osx | winxp-mingw32 | linux64-exp |
|---|---|---|---|---|---|
| | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 |
| ferari | Success | Success | Success | Success | Success |
| fiat | Success | Success | Success | Success | Success |
| ufc | Success | Success | Success | Success | Success |
| instant | Success | Success | Success | Success | Success |
| ufl | Success | Success | Success | Success | Success |
| ffc | Success | Success | Success | Success | Success |
| viper | Success | Success | Success | Success | Success |
| dolfin | Failed | Failed | Failed | Failed | Failed |
| syfi | Success | Success | Success | Success | Success |
| | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 | 8 (8) / 9 |

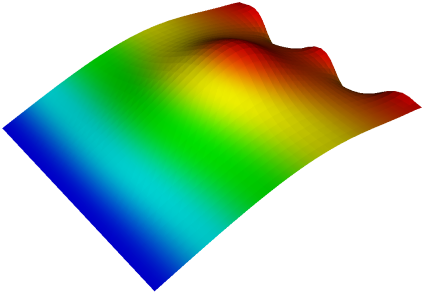
# Examples and Applications

# Poisson's Equation

## Differential equation

$$-\Delta u = f$$

- Heat transfer
- Electrostatics
- Magnetostatics
- Fluid flow
- etc.



# Poisson's Equation

## Variational formulation

Find $u \in V$ such that

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

where

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

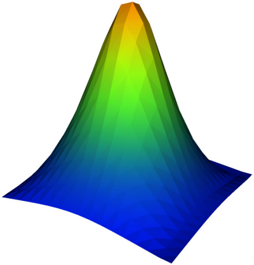
# Poisson's Equation

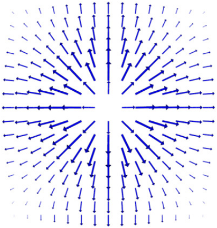
## Implementation

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

u = TrialFunction(V)
v = TestFunction(V)
f = Expression(...)

a = (grad(u), grad(v))
L = (f, v)
```

# Mixed Poisson with $H(\mathrm{div})$ Elements

## Differential equation





$$\sigma + \nabla u = 0$$
$$\nabla \cdot \sigma = f$$

- $u \in L^2$
- $\sigma \in H(\mathrm{div})$

# Mixed Poisson with $H(\mathrm{div})$ Elements

## Variational formulation

Find $(\sigma, u) \in V$ such that

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

where

$$\begin{aligned} a((\sigma, u), (\tau, v)) &= \langle \sigma, \tau \rangle - \langle u, \nabla \cdot \tau \rangle + \langle \nabla \cdot \sigma, v \rangle \\ L((\tau, v)) &= \langle f, v \rangle \end{aligned}$$

# Mixed Poisson with $H(\mathrm{div})$ Elements

## Implementation

```
BDM1 = FunctionSpace(mesh, "BDM", 1)
DG0 = FunctionSpace(mesh, "DG", 0)

V = BDM1 * DG0

(sigma, u) = TrialFunctions(V)
(tau, v) = TestFunctions(V)

f = Expression(...)

a = (sigma, tau) + (u, -div(tau)) + (div(sigma), v)
L = (f, v)
```

Rognes, Kirby, Logg, *Efficient Assembly of* $H(\mathrm{div})$ *and* $H(\mathrm{curl})$ *Conforming Finite Elements* (2009)

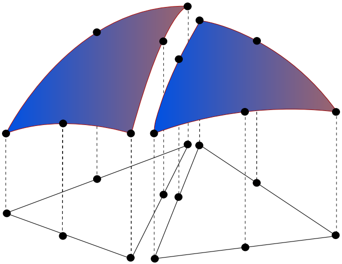
# Poisson's Equation with DG Elements

## 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

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

# Poisson's Equation with DG Elements

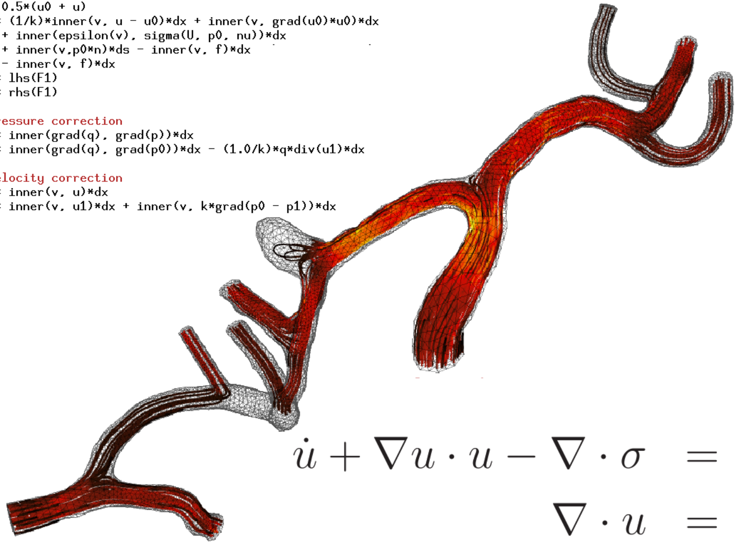
## Implementation

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

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

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

# Computational Hemodynamics

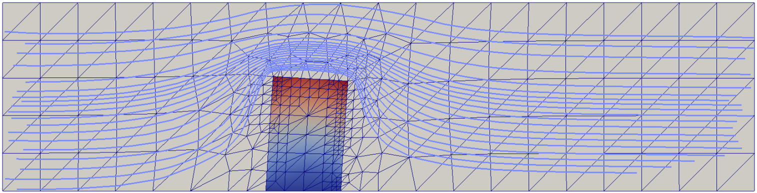
```
# Tentative velocity step
U = 0.5*(u0 + u)
F1 = (1/k)*inner(v, u - u0)*dx + inner(v, grad(u0)*u0)*dx
    + inner(epsilon(v), sigma(U, p0, nu))*dx
    + inner(v,p0*n)*ds - inner(v, f)*dx
    - inner(v, f)*dx
a1 = lhs(F1)
L1 = rhs(F1)

# Pressure correction
a2 = inner(grad(q), grad(p))*dx
L2 = inner(grad(q), grad(p0))*dx - (1.0/k)*q*div(u1)*dx

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



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

Valen-Sendstad, Mardal, Logg, *Simulating the Hemodynamics of the Circle of Willis* (2010)

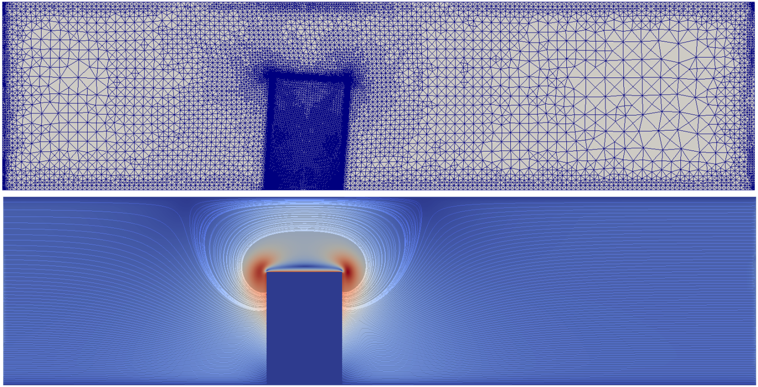
# Fluid–Structure Interaction



- Fluid governed by the incompressible Navier–Stokes equations
- Structure governed by the nonlinear St. Venant–Kirchhoff model
- Mesh and time steps determined adaptively to control the error in a given goal functional to within a given tolerance

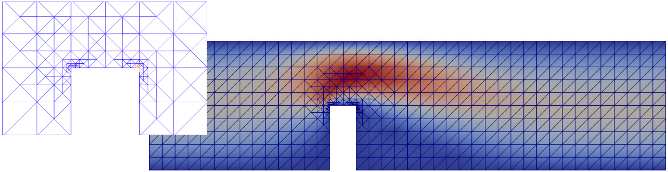
Selim, Logg, Narayanan, Larson, *An Adaptive Finite Element Method for FSI* (2011)

# Adaptive Error Control for FSI



Selim, Logg, Narayanan, Larson, *An Adaptive Finite Element Method for FSI* (2011)

# Adaptive Error Control for Navier–Stokes



```
from dolfin import *

class Noslip(SubDomain): ...

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

# Define test functions and unknown(s)
(v, q) = TestFunctions(V * Q)
w = Function(V * Q)
(u, p) = (as_vector((w[0], w[1])), w[2])

# 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 and pde
M = u[0]*ds(0)
pde = AdaptiveVariationalProblem(F, bcs=[...], M, u=w, ...)

# Compute solution
(u, p) = pde.solve(1.e-4).split()
```

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

**Uniform**

$1.000.000$ dofs, $N$ hours

**Adaptive**

$5.200$ dofs, $127$ seconds

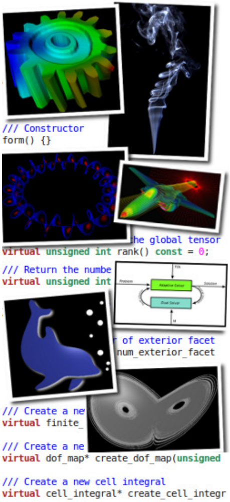
Rognes, Logg, *Automated Goal-Oriented Error Control I* (2010)

# Summary

- Automated solution of differential equations
- Simple installation
- Simple scripting in Python
- Efficiency by automated code generation
- Free/open-source (LGPL)

Upcoming events

- Release of 1.0 (2011)
- Book (2011)
- New web page (2011)
- Mini courses / seminars (2011)




`http://www.fenicsproject.org/`

`http://www.simula.no/research/acdc/`