FEniCS - very short introduction

Jaroslav Hron

Mathematický Ústav UK

UNIVERZITA KARLOVA V PRAZE

matematicko-fyzikální fakulta

FEniCS - http://fenicsproject.org

Automated Solution of Differential Equations by the Finite Element Method

- started in 2003, collaboration between University of Chicago and Chalmers University of Technology
- end of 2011 - version 1.0, tutorial book published (jan 2012) with main contribution by 5 institutions (Simula Research Laboratory, University of Cambridge, University of Chicago, Texas Tech University, KTH Royal Institute of Technology)
- open source license (GNU LGPL v3)
- developement on launchpad (https://launchpad.net/fenics-project)

info:

- FEniCS book - Volume 84 of the Springer Lecture Notes in Computational Science and Engineering
- preliminary version of the book https://launchpad.net/fenics-book
- official tutorial http://fenicsproject.org/documentation/tutorial

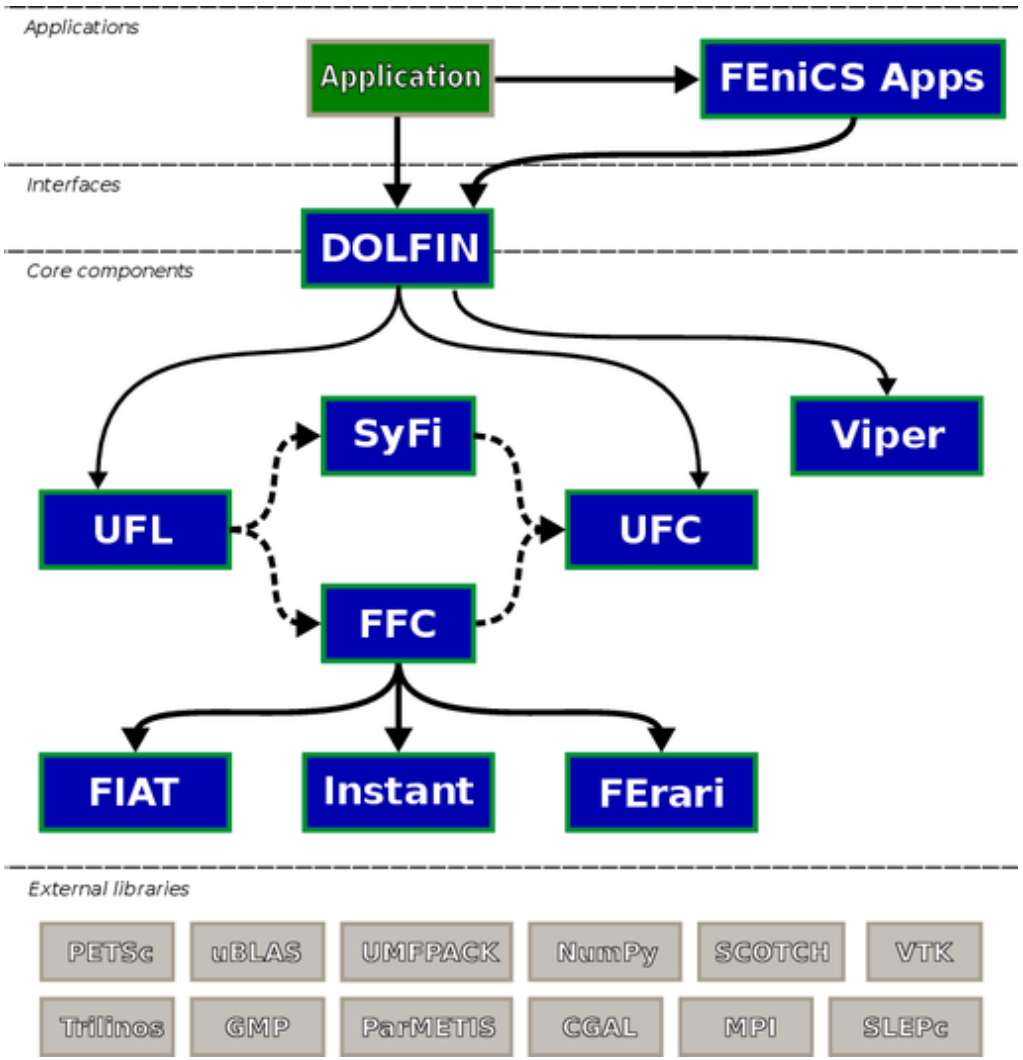
FEniCS - http://fenicsproject.org

- core components:

Dolfin	C++/Python interface of FEniCS, providing a consistent Problem Solving Environment
FFC	FEniCS Form Compiler - compiler for multilinear forms by generating code (C++)
FIAT	FInite element Automatic Tabulator (curently Lagrange, mixed FE)
Instant	Python module that allows for instant inlining of C and C++ code in Python
UFC	Unified Form-assembly Code is a unified framework for finite element assembly
UFL	Unified Form Language is specific language for declaration of finite element discretizations of variational forms

- additional libraries: Dorsal, FErari, SyFi, Viper
- external libraries: PETSc, UMFPACK, Trilinos, CGAL, VTK,....

FEniCS - http://fenicsproject.org



FEniCS - http://fenicsproject.org

$$\operatorname{div}\operatorname{grad} u = f \qquad \text{in } \Omega = [0,1] \times [0,1] \tag{1}$$

$$u = 0 \qquad \text{on } \partial\Omega \tag{2}$$

```
from dolfin import *

mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "Lagrange", 1)

u0 = Constant(0.0)
bc = DirichletBC(V, u0, DomainBoundary())

u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]+x[1]")

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

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

- use iterative solver:

```
prm = parameters['krylov_solver'] # short form
prm['absolute_tolerance'] = 1E-10
prm['relative_tolerance'] = 1E-6
prm['maximum_iterations'] = 1000
set_log_level(PROGRESS)

solve(a == L, u, bc,
      solver_parameters={'linear_solver': 'cg',
                         'preconditioner': 'ilu'})
```

- nonlinear solver

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

F = inner((1.0+u**2)*grad(u), grad(v))*dx - f*v*dx
# jacobian can be computed by ... J  = derivative(F, u_, u)

solve(F == 0 , u, bc ,
      solver_parameters={"newton_solver":
                         {"relative_tolerance": 1e-6}})
```