Building Flexible User Interfaces for Solving PDEs

Anders Logg[*,†] and Garth N. Wells[**]

[*]*Center for Biomedical Computing at Simula Research Laboratory, P.O. Box 134, 1325 Lysaker.*
[†]*Department of Informatics, University of Oslo, Norway.*
[**]*Department of Engineering, University of Cambridge.*

Abstract. FEniCS is a collection of software tools for the automated solution of differential equations by finite element methods. In this note, we describe how FEniCS can be used to solve a simple nonlinear model problem with varying levels of automation. At one extreme, FEniCS provides tools for the fully automated and adaptive solution of nonlinear partial differential equations. At the other extreme, FEniCS provides a range of tools that allow the computational scientist to experiment with novel solution algorithms.

Keywords: PDE, adaptivity, automation, FEniCS
PACS: 02.60.Cb, 02.60.L, 02.70.Dh

INTRODUCTION

The FEniCS Project [1] is a collaborative effort towards the development of innovative concepts and tools for automated scientific computing. Particular emphasis is placed on finite element methods for partial differential equations. An overview of the project can be found in [2]. FEniCS is organized as a collection of interoperable components, all released under a free/open-source license (GNU L/GPL).

One of the main components is DOLFIN which provides the user interface for FEniCS. The user interface is available in two different forms: either as a C++ library or as a Python module. Both provide the user with a range of classes and functions that automate various aspects of the finite element method. A novel feature of FEniCS is that it relies on automated code generation for the general and efficient solution of finite element variational problems. In particular, code for the assembly of finite element matrices and vectors is automatically generated by a form compiler for each particular partial differential equation and each particular choice of finite element discretization. This allows FEniCS to combine two seemingly contradictory properties: generality, by handling a large class of partial differential equations, and efficiency, by generating optimized and highly specific code for each particular equation. For a more detailed exposition, we refer to [3, 4, 5, 6, 7].

In this note, we examine the Python interface of FEniCS/DOLFIN and demonstrate how it can be used to solve a partial differential equation with varying levels of automation. Users may rely on FEniCS to automatically compute solutions of nonlinear partial differential equations with automatic adaptive error control, or use the tools provided by FEniCS to design novel solution algorithms and experiment with different discretizations.

SOLVING A NONLINEAR PDE IN FENICS

As a model problem, we consider the following nonlinear partial differential equation:

$$\begin{array}{rcll} -\nabla\cdot((1+\sin^2 u)\nabla u) &=& \sin(5\pi x), & \text{in } \Omega = (0,1)\times(0,1), \\ u &=& 0 & \text{on } \partial\Omega_D = \{(x,y) : x = 1 \text{ or } y = 1\}, \\ \partial_n u &=& 0 & \text{on } \partial\Omega_N = \partial\Omega \setminus \partial\Omega_D. \end{array} \tag{1}$$

The corresponding variational problem takes the following form: find $u \in V$ such that

$$F(u;v) \equiv \int_\Omega (1+\sin^2 u)\nabla u\cdot\nabla v\,\mathrm{d}x - \int_\Omega \sin(5\pi x)\,v\,\mathrm{d}x = 0$$

for all $v \in V$ where $V = \{v \in H^1(\Omega) : v = 0 \text{ on } \Omega_D\}$.

Automated discretization, solution, and error control

We first demonstrate how the solution of (1) can be computed using fully automated discretization, solution, and goal-oriented error control. Automated goal-oriented error control was recently added to FEniCS (in DOLFIN 0.9.8), based on an automation [8] of the framework for goal-oriented error control presented in [9, 10]. To compute the solution with automatic error control, we specify a functional of interest (the goal functional) $\mathscr{M} : V \to \mathbb{R}$. Automated goal-oriented error control then seeks to find a suitable discrete subspace $V_h \subset V$ such that

$$|\mathscr{M}(u) - \mathscr{M}(u_h)| \leq \varepsilon,$$

where $\varepsilon > 0$ is a given tolerance and $u_h \in V_h$ is the finite element solution of the variational problem on V_h. Here, we consider the simple case where the functional is given by the average of the solution over the domain; that is,

$$\mathscr{M}(u) = \int_\Omega u \, \mathrm{d}x.$$

To solve (1) to within a given tolerance in FEniCS, one only needs to define the computational domain, the variational problem, the boundary conditions and the goal functional, and then ask FEniCS to compute the solution. This is illustrated below where we list a complete program for computing the solution with fully automated adaptive error control. The solution to this problem is shown in Figure 1, along with the adaptively refined mesh.

```python
from dolfin import *

# Create mesh and function space
mesh = UnitSquare(4, 4)
V = FunctionSpace(mesh, "CG", 1)

# Define Dirichlet boundary
def dirichlet_boundary(x):
    return x[0] == 1.0 or x[1] == 1.0

# Define boundary condition
g = Constant(0)
bc = DirichletBC(V, g, dirichlet_boundary)

# Define source term
f = Expression("sin(5*pi*x[0])")

# Define variational problem
u = Function(V)
v = TestFunction(V)
F = inner((1 + sin(u)**2)*grad(u), grad(v))*dx - f*v*dx
M = u*dx

# Solve variational problem
problem = AdaptiveVariationalProblem(F, bcs=[bc], goal_functional=M, u=u)
u = problem.solve(1e-5)

# Plot and save solution
plot(u, title="Solution", interactive=True)
file = File("solution.pvd")
file << u
```

Automated discretization and solution

A user may wish to rely on FEniCS for automated discretization and solution, but design a novel adaptive algorithm or compute the solution on a fixed mesh. We illustrate this in the program listed below. Here, `AdaptiveVariationalProblem` is replaced by `VariationalProblem`. These two classes differ in that `VariationalProblem` expects as input the nonlinear variational problem $F(u;v)$ as well as the its Fréchet derivative F'. This may either be supplied by the user or, as in the code listed below, computed automatically by

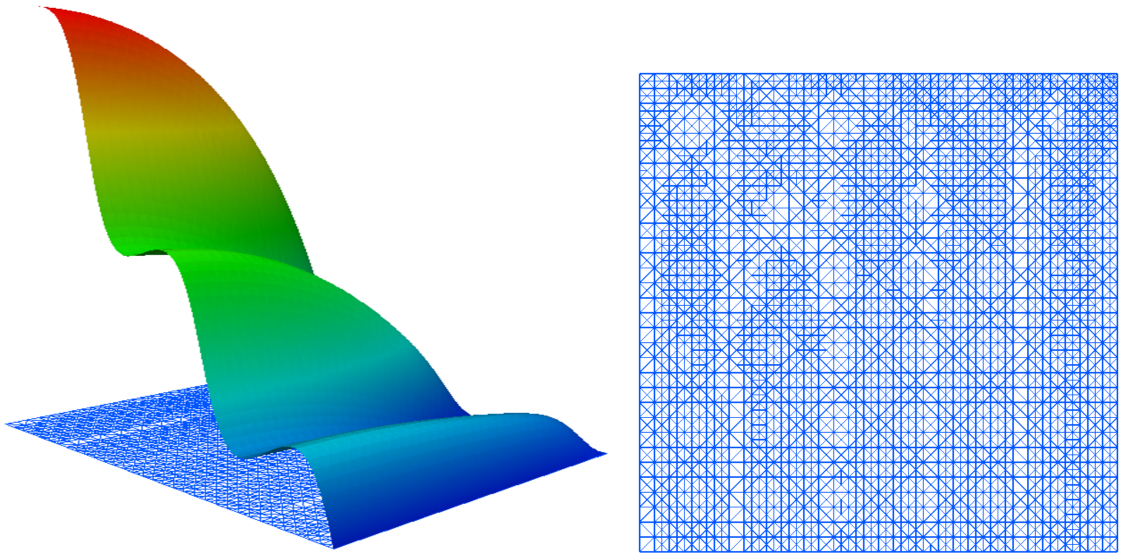

FIGURE 1. The computed approximate solution u_h of (1) and the adaptive refined mesh.

FEniCS by a call to `derivative`. Both `AdaptiveVariationalProblem` and `VariationalProblem` use Newton's method to solve nonlinear systems. The automatic differentiation is symbolic, and not numerical. For an overview of automatic differentiation in FEniCS, we refer to [11].

```
[initial part of program same as above]

# Define variational problem
u  = Function(V)
v  = TestFunction(V)
F  = inner((1 + sin(u)**2)*grad(u), grad(v))*dx - f*v*dx
du = TrialFunction(V)
dF = derivative(F, u, du)

# Solve variational problem
problem = VariationalProblem(dF, F, bcs=[bc], nonlinear=True)
problem.solve(u)

# Plot and save solution
plot(u, title="Solution", interactive=True)
file = File("solution.pvd")
file << u
```

Automated discretization

Although FEniCS provides tools for simple and automatic solution of linear and nonlinear partial differential equations, it is recognized that solution methods are diverse and evolving. FEniCS therefore provides a set of tools that can be used to experiment with different solution techniques. Two such tools are the `assemble` and `solve` functions. In the code snippet below, we illustrate how these may be used to assemble and solve a linear system for a single iteration of Newton's method for the solution of the nonlinear partial differential equation (1). Optional arguments may be given to the `solve` function to select different direct or iterative linear solution methods and, in the latter case, to select a preconditioner.

```
# Assemble linear system
A = assemble(dF)
b = assemble(F)
```

```
# Apply boundary conditions
bc.apply(A, b)

# Solve linear system
dx = Vector()
solve(A, dx, b)
```

Low-level interaction

FEniCS also provides low-level interfaces that allow advanced users to fine-tune solution algorithms. One such interface is the UFC interface [12] which specifies the mechanisms for communication between FEniCS and application-specific generated code. This code is normally generated automatically, but users with specific needs may also choose to implement this code, or parts of it, manually and communicate the code to FEniCS through the UFC interface.

DISCUSSION AND CONCLUSIONS

We have demonstrated how FEniCS can be used to solve partial differential equations with varying level of automation and complexity. The user interface of FEniCS is the result of an iterative development process over many years. With the recent release of DOLFIN 0.9.8, FEniCS is approaching the release of a stable interface with the expected release of DOLFIN 1.0 at the end of 2010.

ACKNOWLEDGMENTS

This work is supported by an Outstanding Young Investigator grant from the Research Council of Norway, NFR 180450. This work is also supported by a Center of Excellence grant from the Research Council of Norway to the Center for Biomedical Computing at Simula Research Laboratory. We acknowledge the work of many people that have contributed to the development of FEniCS.

REFERENCES

1. The FEniCS project (http://www.fenics.org) (2010), URL `http://www.fenics.org`.
2. A. Logg, *Arch. Comput. Methods Eng.* **14**, 93–138 (2007), ISSN 1134-3060.
3. A. Logg, and G. N. Wells, *ACM Transactions on Mathematical Software* **32**, 1–28 (2010).
4. R. C. Kirby, and A. Logg, *ACM Transactions on Mathematical Software* **32**, 417–444 (2006), ISSN 0098-3500.
5. R. C. Kirby, and A. Logg, *ACM Transactions on Mathematical Software* **33** (2007), ISSN 0098-3500.
6. K. B. Ølgaard, and G. N. Wells, *ACM Transactions on Mathematical Software* **37**, 8:1–8:23 (2010).
7. M. S. Alnæs, and K.-A. Mardal, *Transactions on Mathematical Software* **37** (2010).
8. M. E. Rognes, and A. Logg, *SIAM J. Sci. Comput.* (2010).
9. K. Eriksson, D. Estep, P. Hansbo, and C. Johnson, *Acta Numerica* **4**, 105–158 (1995).
10. R. Becker, and R. Rannacher, *Acta Numerica* **10**, 1–102 (2001).
11. M. S. Alnæs, *A Compiler Framework for Automatic Linearization and Efficient Discretization of Nonlinear Partial Differential Equations*, PhD thesis, University of Oslo, Unipub, Oslo, Norway (2009), iSSN 1501-7710, No. 884.
12. M. S. Alnæs, A. Logg, K.-A. Mardal, O. Skavhaug, and H. P. Langtangen, *International Journal of Computational Science and Engineering* **4**, 231–244 (2009).