PAMM · Proc. Appl. Math. Mech. **7**, 1010601–1010602 (2007) / **DOI** 10.1002/pamm.200700234

Automated solution of differential equations

Anders Logg[1,*]

[1] Simula Research Laboratory, Martin Linges v 17, Fornebu, PO Box 134, 1325 Lysaker, Norway.

Differential equations are solved routinely by large computer programs, but the solution process is rarely automated. Each equation requires a different program and each such program requires a considerable amount of work to develop and maintain.

The FEniCS project provides a set of tools that automate important aspects of the solution process, ultimately aiming at a complete automation of computational mathematical modeling, including the automation of discretization, discrete solution, error control, modeling and optimization. A key component of FEniCS is the FEniCS Form Compiler (FFC), which automates the discretization of differential equations by taking as input a variational problem in mathematical notation and generating highly efficient optimized low-level code for the evaluation of the corresponding discrete operator.

1 Introduction

The finite element method provides a general framework for solving differential equations and thus, by automating the finite element method, one may automate the solution of differential equations. However, a complete automation of the finite element remains an outstanding challenge. A further challenge is to design and implement such an automating system so that it becomes both *general* and *efficient*. The FEniCS project [1] solves this problem by *code generation*. Automated code generation allows a system to be both general and *specific* (and thereby efficient) since code may be generated specifically for any given input (differential equation).

2 Finite element code generation

In writing software for solving differential equations by the finite element method, one may identify common tasks that may be implemented as reusable components. Such components may be data structures and algorithms for computational meshes, linear algebra and plotting. However, other components must be implemented specifically for each specific differential equation. In particular, this is true for the computation of the discrete operator (stiffness matrix) arising from the finite element discretization of a differential operator. Thus, one may implement a finite element program for solving a particular differential equation by combining reusable components from a library with special purpose code. Writing the special purpose code for computing the discrete operator is often time consuming and prone to errors. However, as demonstrated in [2–5], the special purpose code may be generated automatically from a high-level description of the differential operator. Thus, by combining reusable components for algorithms on computational meshes and linear algebra with automated code generation for the computation of the discrete operator, one may build an automated system for the solution of differential equations. These concepts are illustrated in Figure 1.

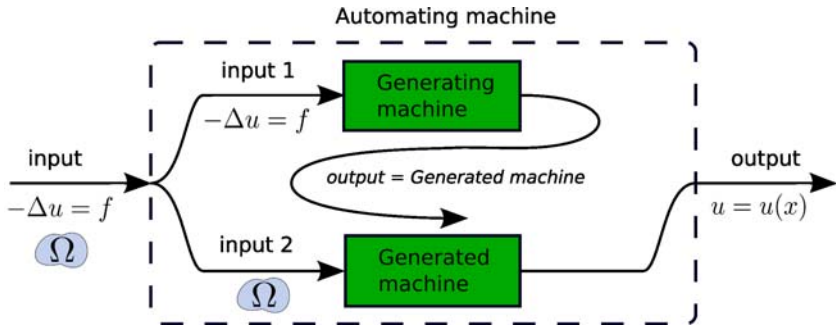

Fig. 1 Finite element code generation. A specialized program is generated for a subset of the input (the differential equation). The solution is then computed by running the generated program for the remaining input (mesh, coefficients and other parameters).

The FEniCS project is organized as a collection of interoperable components for automated solution of differential equations. Some of these components provide reusable libraries for mesh data structures and algorithms. Other components, like

* Corresponding author E-mail: logg@simula.no, Phone: +47 678 28288

the *form compilers* FFC and SyFi provide tools for finite element code generation. A form compiler accepts as input a finite element variational form and generates as output efficient low-level code for assembly of the corresponding discrete operator. Recently, we have developed a specification for a fixed interface adhered to by the generated code. The interface is named UFC (Unified Form-assembly Code) and is described in detail in [6]. A fixed and documented interface makes it possible to use the code generated by the form compilers FFC and SyFi in other projects (that support the UFC specification). Similarly, one may hook up UFC code generated by other form compilers or hand-written UFC code to FEniCS. With the output from form compilers fixed, we plan to turn our attention to the specification of a common input language for form compilers (UFL, Unified Form Language).

On top of the collection of low-level library components and code generation tools discussed above, the FEniCS project provides a common high-level interface to these components (DOLFIN). The interface is available both as a C++ library and as a Python module. Recently, support has been added to DOLFIN for just-in-time compilation of variational forms, meaning that finite element variational forms may be specified and evaluated from within a Python script; code is generated at run-time (by a form compiler) and compiled (by a C++ compiler).

3 Generality and efficiency

By code generation, one may obtain a system that is both general and efficient. FEniCS currently supports code generation for arbitrary order continuous and discontinuous Lagrange elements (standard piecewise polynomials), Brezzi–Douglas–Marini elements, Raviart–Thomas elements, Nedelec elements, Crouzeix–Raviart elements and arbitrary mixed elements. We don't claim that the code generation is optimal or close to optimal for all combinations of finite elements and variational forms. However, in [4], speedups of several orders of magnitude are demonstrated for a set of standard test cases. In [4], we also discuss the limitations and shortcomings of our current code generation approach.

4 Examples

As an example, we consider here the implementation of a solver for Poisson's equation, $-\Delta u = f$. We consider the following standard variational problem: Find $u \in V_h$ such that $a(v, u) = L(v)$ for all $v \in V_h$, where $a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x$, $L(v) = \int_\Omega v f \, \mathrm{d}x$ and V_h is the space of piecewise linear functions. We list the Python implementation below. For brevity, we have left out the definition of the right-hand side f and the boundary condition.

```
from dolfin import *

mesh = UnitSquare(32, 32)
element = FiniteElement("Lagrange", "triangle", 1)

v = TestFunction(element)
u = TrialFunction(element)
a = dot(grad(v), grad(u))*dx
L = v*f*dx

pde = LinearPDE(a, L, mesh, bc)
u = pde.solve()
plot(u)
```

Acknowledgements The FEniCS project [1] was initiated in 2003 to provide free software for the automation of computational mathematical modeling [7]. FEniCS is a joint project between the University of Chicago, Argonne National Laboratory, Delft University of Technology, Royal Institute of Technology KTH, Simula Research Laboratory, Finite Element Center and University of Cambridge (in order of appearance). Many people have made important contributions, including Martin Sandve Alnæs, Johan Hoffman, Johan Jansson, Claes Johnson, Robert C. Kirby, Matthew G. Knepley, Hans Petter Langtangen, Kent-Andre Mardal, Kristian Oelgaard, Marie Rognes, L. Ridgway Scott, Ola Skavhaug, Andy R. Terrel, Garth N. Wells, Åsmund Ødegård and Magnus Vikstrøm. FEniCS is free software licensed under the GNU GPL or LGPL licenses. To download the latest release of FEniCS, visit `http://www.fenics.org/`.

References

[1] FEniCS, The FEniCS Project, 2007, URL: `http://www.fenics.org/`.
[2] R. C. Kirby, M. G. Knepley, A. Logg, and L. R. Scott, SIAM J. Sci. Comput. **27**(3), 741–758 (2005).
[3] R. C. Kirby, A. Logg, L. R. Scott, and A. R. Terrel, SIAM J. Sci. Comput. **28**(1), 224–240 (2006).
[4] R. C. Kirby and A. Logg, ACM Transactions on Mathematical Software **32**(3), 417–444 (2006).
[5] R. C. Kirby and A. Logg, ACM Transactions on Mathematical Software **33**(3) (2007).
[6] M. Alnæs, H. P. Langtangen, A.Logg, K. A. Mardal, and O. Skavhaug, UFC Specification and User Manual, 2007, URL: `http://www.fenics.org/ufc/`.
[7] A. Logg, Automation of Computational Mathematical Modeling, PhD thesis, Chalmers University of Technology, Sweden, 2004.