Advances in Water Resources 34 (2011) 1082–1101

Contents lists available at ScienceDirect

Advances in Water Resources

journal homepage: www.elsevier.com/locate/advwatres

A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier–Stokes equations

Mikael Mortensen [a,b], Hans Petter Langtangen [b,c,*], Garth N. Wells [d]

[a] Norwegian Defence Research Establishment, 2007 Kjeller, Norway
[b] Center for Biomedical Computing, Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway
[c] Department of Informatics, University of Oslo, P.O. Box 1080, Blindern, 0316 Oslo, Norway
[d] Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, United Kingdom

ARTICLE INFO

Article history:
Available online 22 March 2011

Keywords:
Turbulent flow
RANS models
Finite elements
Python
Object-oriented programming
Problem solving environment

ABSTRACT

Finding an appropriate turbulence model for a given flow case usually calls for extensive experimentation with both models and numerical solution methods. This work presents the design and implementation of a flexible, programmable software framework for assisting with numerical experiments in computational turbulence. The framework targets Reynolds-averaged Navier–Stokes models, discretized by finite element methods. The novel implementation makes use of Python and the FEniCS package, the combination of which leads to compact and reusable code, where model- and solver-specific code resemble closely the mathematical formulation of equations and algorithms. The presented ideas and programming techniques are also applicable to other fields that involve systems of nonlinear partial differential equations. We demonstrate the framework in two applications and investigate the impact of various linearizations on the convergence properties of nonlinear solvers for a Reynolds-averaged Navier–Stokes model.

1. Introduction

Turbulence is the rule rather than the exception when water flows in nature, but finding the proper turbulence model for a given flow case is demanding. There exists a large number of different turbulence models, and a researcher in computational turbulence would benefit from being able to easily switch between models, combine models, refine models and implement new ones. As the models consist of complex, highly nonlinear systems of partial differential equations (PDEs), coupled with the Navier–Stokes (NS) equations, constructing efficient and robust iteration techniques is model- and problem-dependent, and hence subject to extensive experimentation. Flexible software tools can greatly assist the researcher experimenting with models and numerical methods. This work demonstrates how flexible software can be designed and implemented using modern programming tools and techniques.

Precise prediction of turbulent flows is still a very challenging task. It is commonly accepted that solutions of the Navier–Stokes equations, with sufficient resolution of all scales in space and time (Direct Numerical Simulation, DNS), describe turbulent flow. Such an approach is, nevertheless, computationally feasible only for low Reynolds number flow and simple geometries, at least for the foreseeable future. Large Eddy Simulations (LES), which resolve large scale motions and use subgrid models to represent the unresolved scales are computationally less expensive than DNS, but are still too expensive for the simulation of turbulent flows in many practical applications. A computationally efficient approach to turbulent flows is to work with Reynolds-averaged Navier–Stokes (RANS) models. RANS models involve solving the incompressible NS equations in combination with a set of transport equations for statistical turbulence quantities. The uncertainty in RANS models lies in the extra transport equations, and for a given flow problem it is a challenge to pick an appropriate model. There is hence a need for a researcher to experiment with different models to arrive at firm conclusions on the physics of a problem.

Most commercial computational fluid dynamics (CFD) packages contain a limited number of turbulence models, but allow users to add new models through "user subroutines" which are called at each time level in a simulation. The implementation of such routines can be difficult, and new models might not fit easily within the constraints imposed by the design of the package and the "user subroutine" interface. The result is that a specific package may only support a fraction of the models that a practitioner would wish to have access to. There is a need for CFD software with a flexible design so that new PDEs can be added quickly and reliably, and so that solution approaches can easily be composed. We believe that the most effective way of realizing such features is to have a *programmable* framework, where the models and numerics are

* Corresponding author at: Center for Biomedical Computing, Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway.
E-mail addresses: mikael.mortensen@ffi.no (M. Mortensen), hpl@simula.no (H.P. Langtangen), gnw20@cam.ac.uk (G.N. Wells).

0309-1708/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advwatres.2011.02.013

defined in terms of a compact, high-level computer language with a syntax that is based on mathematical language and abstractions.

A software system for RANS modeling must provide higher-order spatial discretizations, fine-grained control of linearizations, support for both Picard and Newton type iteration methods, under-relaxation, restart of models, combinations of models and the easy implementation of new PDEs. Standard building blocks needed in PDE software, such as forming coefficient matrices and solving linear systems, can act as black boxes for a researcher in computational turbulence. To the authors' knowledge, there is little software with the aforementioned flexibility for incompressible CFD. There are, however, many programmable environments for solving PDEs. A non-exhaustive list includes Cactus [7], COMSOL Multiphysics [11], deal.II [12,2], Diffpack [13], DUNE [16], FEniCS [19] [37], Dular and Geuzaine [15], GetFEM++ [20], OpenFOAM [43], Overture [44], Proteus [47] and SAMRAI [51]. Only a few of these packages have been extensively used for turbulent flow. OpenFOAM [43] is a well-structured and widely used object-oriented C++ framework for CFD, based on finite volume methods, where new models can quite easily be added through high-level C++ statements. Overture [44,6] is also an object-oriented C++ library used for CFD problems, allowing complex movements of overlapping grids. Proteus [47] is a modern Python- and finite element-based software environment for solving PDEs, and has been used extensively for CFD problems, including free surface flow and RANS modeling. FEniCS [19,36] is a recent C++/Python framework, where systems of PDEs and corresponding discretization and iteration strategies can be defined in terms of a few high-level Python statements which inherit the mathematical structure of the problem, and from which low level code is generated. The approach advocated in this work utilizes FEniCS tools. All FEniCS components are freely available under GNU general public licenses [19]. A number of application libraries that make use of the FEniCS software have been published [57]. For instance, `cbc.solve` [9] is a framework for solving the incompressible Navier–Stokes equations and the Rheology Application Engine (Rheagen) [48] is a framework for simulating non-Newtonian flows. Both applications share some of the features of the current work.

Traditional simulation software packages are usually implemented in Fortran, C, or C++ because of the need for high computational performance. A consequence is that these packages are less user-friendly and flexible, but far more efficient, than similar projects implemented in scripting languages such as Matlab or Python. In FEniCS, scripting is combined with symbolic mathematics and code generation to provide both user-friendliness and efficiency. Specifically, the Unified Form Language (UFL), a domain-specific language for the specification of variational formulations of PDEs, is embedded within the programming language Python. Variational formulations are then just-in-time compiled into C++ code for efficiency. The generated C++ code can be expected to outperform hand-written quadrature code since special-purpose PDE compilers [1,28,42] are employed. UFL has built-in support for automatic differentiation, derivation of adjoint equations, etc., which makes it particularly useful for complicated and coupled PDE problems.

Several authors have addressed how object-oriented and generative programming can be used to create flexible libraries for solving PDEs, but there are significantly fewer contributions dealing with the design of frameworks on top of such libraries for addressing multi-physics problems and coupling of PDEs [45,31,33,23,54,40,50]. These contributions focus on how the C++ or Fortran 90 languages can be utilized to solve such classes of problems. This work builds on these cited works, but applies Python as programming language and FEniCS as tool for solving PDEs. Python has strong support for dynamic classes and object orientation, and since variables are not declared in Python, generative programming comes without any extra syntax (in contrast with templates in C++). Presented code examples from the framework will demonstrate how these features, in combination with FEniCS, result in clean and compact code, where the specification of PDE models and linearization strategies can be expressed in a mathematical syntax.

FEniCS supports finite element schemes, including discontinuous Galerkin methods [41], but not finite difference methods. Many finite volume methods can be constructed as low-order discontinuous Galerkin methods using FEniCS [55]. Despite the development of several successful methods for solving the NS equations and LES models by finite element methods, finite element methods have not often been applied to RANS models, though some research contributions exist in this area [21,3,52,38].

The remainder of this paper is organized as follows: Section 2 demonstrates the use of FEniCS for solving simple PDEs and briefly elaborates some key aspects of FEniCS. Section 3 presents a selection of PDEs which form the basis of some common RANS models. Finite element formulations of a typical RANS model and the iteration strategies for handling nonlinear equations appear in Section 4. The software framework for NS solvers and RANS models is described in Section 5. Section 6 demonstrates two applications of the framework and investigates the impact of different types of linearizations. In Section 7 we briefly discuss the computational efficiency of the framework, and some concluding perspectives are drawn in Section 8. The code framework we describe, cbc.rans, is open source and available under the Lesser GNU Public license [8].

2. FEniCS for solving differential equations

FEniCS is a collection of software tools for the automated solution of differential equations by finite element methods. FEniCS includes tools for working with computational meshes, linear algebra and finite element variational formulations of PDEs. In addition, FEniCS provides a collection of ready-made solvers for a variety of partial differential equations.

2.1. Solving a partial differential equation

To illustrate how PDEs can be solved in FEniCS, we consider the weighted Poisson equation $-\nabla\cdot(\kappa\nabla u)=f$ in some domain $\Omega\subset\mathbb{R}^d$ with $\kappa=\kappa(x)$ a given coefficient. On a subset of the boundary, denoted by $\partial\Omega_D$, we prescribe a Dirichlet condition $u=0$, while on the remainder of the boundary, denoted by $\partial\Omega_R$, we prescribe a Robin condition $-\kappa\partial u/\partial n=\alpha(u-u_0)$, where α and u_0 are given constants.

To solve the above boundary-value problem, we first need to define the corresponding variational problem. It reads: find $u\in V$ such that

$$F \equiv \int_\Omega \kappa\nabla u\cdot\nabla v\,dx-\int_\Omega fv\,dx+\int_{\partial\Omega_R}\alpha(u-u_0)v\,ds=0\quad\forall v\in V, \tag{1}$$

where V is the standard Sobolev space $H^1(\Omega)$ with $u=v=0$ on $\partial\Omega_D$. The function u is known as a trial function and v is known as a test function. We can partition F into a "left-hand side" $a(u,v)$ and a "right-hand side" $L(v)$,

$$F=a(u,v)-L(v), \tag{2}$$

where

$$a(u,v)=\int_\Omega\kappa\nabla u\cdot\nabla v\,dx+\int_{\partial\Omega_R}\alpha uv\,ds, \tag{3}$$

$$L(v)=\int_\Omega fv\,dx+\int_{\partial\Omega_R}\alpha u_0 v\,ds. \tag{4}$$