Implementing a k-epsilon Turbulence Model in the FEniCS Finite Element Programming Environment

Conference Paper · October 2013

CITATIONS
4

READS
6,392

5 authors, including:

Kristian Valen-Sendstad
Simula Research Laboratory
71 PUBLICATIONS **1,704** CITATIONS
SEE PROFILE

Mikael Mortensen
University of Oslo
78 PUBLICATIONS **1,298** CITATIONS
SEE PROFILE

Hans Petter Langtangen
Simula Research Laboratory
380 PUBLICATIONS **4,082** CITATIONS
SEE PROFILE

Implementing a k-ε Turbulence Model in the FEniCS Finite Element Programming Environment

Kristian Valen-Sendstad[1,3]
[1]Simula School of Research and Innovation
e–mail: kvs@simula.no

Mikael Mortensen[2], **Hans Petter Langtangen**[3,4] **and Bjørn Anders Pettersson Reif**[2,5]
Kent-André Mardal[3,4]

[2]Norwegian Defence Research Establishment (FFI)

[3]Simula Research Laboratory
Center for Biomedical Computing

[4]University of Oslo
Department of Informatics

[5]University of Oslo
Department of Mathematics

Summary The purpose of the present paper is to explore how a finite element method framework, such as FEniCS, can be used to implement a common k-ε model. The reason is that a flexible, high-level programming environment is needed to experiment with the numerical methods that suits the turbulence model. Hopefully, this will reveal whether more sophisticated models, will be easy to add, and if there are technical reasons for why the finite element method should not be used for RANS modelling.

Introduction

Precise prediction of turbulent flows remains to be an extremely challenging computational problem. Although solution of the Navier-Stokes equations with sufficient resolution of all scales in space and time (Direct Numerical Simulation, DNS) provides, in principle, an accurate approach to turbulence, the approach is still only feasible for low Reynolds number flow in simple geometries. Coarser meshes with subgrid models for the unresolved scales, as in Large Eddy Simulation (LES), is a computationally more efficient approach, but not efficient enough for making Computational Fluid Dynamics (CFD) a useful tool for turbulent flows in applied sciences and engineering practice. The most common strategy is to work with Reynolds Averaged Navier-Stokes (RANS) models, which basically implies solving the incompressible Navier-Stokes equations with an extra set of transport equations for turbulent quantities. There is a wealth of RANS models around, and for a researcher in computational turbulence, there is a need for experimenting with many different models in a flexible way. How to achieve such flexibility is the topic of a research project for which the present paper represents a first step.

Traditionally, CFD packages come with support of a few turbulence models only, but with possibilities of adding new models, usually through a ”user subroutine” that is called at each time step. The associated implementation may be cumbersome, and new models might not easily fit with the constraints of the ”user subroutine”. This calls for CFD software with a flexible design for implementing new partial differential equations (PDEs). There is surprisingly little software of this type around. Many programmable frameworks for PDEs are available, but

few can show a solid position in CFD and strong support for solving systems of Navier-Stokes and transport equations in various ways. OpenFOAM [4] is a C++ framework for CFD, based on finite volume methods, where it is quite easy to add new models through a few high-level C++ statements. FEniCS [2] is an upcoming suite of computing components, based on finite element discretizations in space, that seems to offer the flexibility needed in computational turbulence. The purpose of the present paper is to explore how FEniCS can be used to implement the common k-ε model and if other, more sophisticated models, will be easy to add.

A software system supporting RANS modeling must have a set of particular features. Some models need higher-order discretization in space, and perhaps also in time (though splitting of the PDE systems, which is very common, in itself prevents more than second-order accuracy in time). Since the systems of PDEs are severely nonlinear, the degree of implicitness in the numerical solution strategies must be controlled. That is, one must be able to freely cluster PDEs to form a system that is solved in a fully implicit way. Moreover, one must be able to iterate between such systems to solve the overall system of PDEs in the complete model. Since the nonlinearities may be severe in turbulence models, causing iterations to diverge, there will be need for adding a family of under relaxation parameters and relax quantities in updates. Also certain difficult nonlinear terms in the equations may need relaxation parameters. Tailoring the iteration strategies in this way is much more straightforward if the researcher can program directly, but the program must be compact, with a syntax close to the mathematical formulation of the algorithm to enhance overview and avoid programming errors. Finally, it must be easy to add new PDEs and constitutive relations, and combine PDEs in flexible ways to form new models. RANS modeling is, in these regards, a particularly challenging area for programmable PDE frameworks.

One way of attacking the required flexibility outlined in the previous paragraph is to implement a model as a fairly short, high-level program where all the key numerical steps are visible and can be freely adjusted by the programmer. Comprehensive building blocks, such as forming coefficient matrices and solving linear systems, can act as black boxes for the researcher in computational turbulence. However, the composition of PDEs and constitutive relations, choice of explicit versus implicit handling of terms, operator splitting in time, and nonlinear iteration strategies are details of which the researcher wants to have full control. We think FEniCS is a framework which shows great promise in this context.

The finite volume method is the dominating discretization strategy for fluid dynamics problems, although finite elements are successfully used for solving the Navier-Stokes equations by many groups [1]. However, finite elements have to a considerably less extent been successfully applied to RANS models, for which the finite volume method is totally dominating (c.f. [3]). One goal of the present paper is therefore to explore finite elements in the context of a k-ε model and compare the results with regular finite volume implementations. As already mentioned, FEniCS offers solely finite elements for the spatial discretization.

The paper is organized as follows. First, we provide a brief description of FEniCS. Then we present the mathematics of the particular k-ε formulation we use in this paper and the corresponding finite element formulation. Examples of the FEniCS implementation is given and we run a simple test case for verifying the implementation.

About FEniCS

FEniCS is free software for automated solution of differential equations. Within the FEniCS project there are software tools for working with computational meshes, finite element varia-

tional formulations of PDEs, ODE solvers and linear algebra. The vision of FEniCS is "to set a new standard in Computational Mathematical Modeling (CMM), which is the Automation of CMM, towards the goals of generality, efficiency, and simplicity, concerning mathematical methodology, implementation, and application" [2].

FEniCS consists of several stand-alone software modules which together form a programming environment for solving PDEs. Solution of PDEs in FEniCS can be accomplished by writing a fairly short C++ or Python program for defining the problem-specific input to a problem. This program calls up general libraries containing numerical algorithms that are common to a wide range of problems (e.g., computation of finite elements, assembly of global matrices, iterative solvers for linear systems, preconditioners, etc.).

The Reynolds Averaged Navier-Stokes Equations

Turbulent flows are described by the Navier-Stokes equations. For an incompressible flow with constant density $\rho(\boldsymbol{x}, t)$ these equation include the momentum and continuity equations that take the form

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u} = -\frac{1}{\rho} \boldsymbol{\nabla} p + \nu \boldsymbol{\nabla}^2 \boldsymbol{u} + \boldsymbol{f}, \tag{1}$$

and

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u}(\boldsymbol{x}, t)$ is the instantaneous velocity vector, $p(\boldsymbol{x}, t)$ is pressure, ν is the kinematic viscosity and $\boldsymbol{f}$ represents body forces. Here and throughout this paper the boldface font is used to indicate a vector, whereas scalar quantities are represented with a regular font. In statistical modeling of turbulent flows the instantaneous velocity and pressure are decomposed into mean and fluctuating parts as $u_i = U_i + u_i'$ and $p = P + p'$, where u_i is a component of the velocity vector, the capital letters indicate a mean and prime indicates a fluctuating quantity. For Reynolds Averaged Navier-Stokes (RANS) modelling U_i is an ensemble average that represents the first moment of u_i. By introducing this decomposition into (1) and averaging, the transport equation for each component of the average velocity vector can be found from

$$\frac{\partial U_i}{\partial t} + \boldsymbol{U} \cdot \boldsymbol{\nabla} U_i = -\frac{1}{\rho} \frac{\partial P}{\partial x_i} + \nu \boldsymbol{\nabla}^2 U_i - \boldsymbol{\nabla} \cdot \overline{\boldsymbol{u}' u_i'}. \tag{3}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{U} = 0, \tag{4}$$

The k-ε turbulence model

Turbulence modelling is a matter of expression the unknown quantity $\overline{\boldsymbol{u}_i' u_j'}$ (the overbar represents an ensemble average) in terms of resolved quantities like $\boldsymbol{U}$ and $\boldsymbol{\nabla U}$, or by additional quantities for turbulence length- and timescales, which are then governed by additional transport equations.

The k-ε model is the most widely used general purpose turbulence model. There is an enormous number of variations of the model, but the accepted 'standard' k-ε model refers to the form described by Jones & Launder [6] with empirical constants taken from Launder & Sharma [7]. The standard k-ε model uses wallfunctions instead of regular boundary conditions and does not integrate the governing equations all the way up to solid walls. The wallfunctions represent an additional complexity to turbulence modelling that we are not willing to deal with at the current stage. For this reason the model that is to be used throughout this work is the more straightforward low-Reynolds number version of Launder & Sharma [7] that is integrated all

the way up to solid walls where regular Dirichlet boundary conditions apply. The model is summarized by equations (5)-(12):

$$\frac{\partial k}{\partial t} + \boldsymbol{U} \cdot \boldsymbol{\nabla} k = \boldsymbol{\nabla} \cdot \left[\left(\nu + \frac{\nu_T}{\sigma_k} \right) \boldsymbol{\nabla} k \right] + P_k - \varepsilon - 2\nu \left\| \boldsymbol{\nabla} \sqrt{k} \right\|^2, \tag{5}$$

$$\frac{\partial \varepsilon}{\partial t} + \boldsymbol{U} \cdot \boldsymbol{\nabla} \varepsilon = \boldsymbol{\nabla} \cdot \left[\left(\nu + \frac{\nu_T}{\sigma_\varepsilon} \right) \boldsymbol{\nabla} \varepsilon \right] + \left(C_{\varepsilon 1} P_k - f_2 C_{\varepsilon 2} \varepsilon \right) \frac{\varepsilon}{k} + 2\nu\nu_T \left\| \boldsymbol{\nabla}^2 \boldsymbol{U} \right\|^2, \tag{6}$$

$$\overline{u_i' u_j'} = -\nu_T \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) + \frac{2}{3} k \delta_{ij} \tag{7}$$

$$\nu_T = C_\mu f_\mu k^2 / \varepsilon, \tag{8}$$

$$P_k = -\overline{u_i' u_j'} \frac{\partial U_i}{\partial x_j}, \tag{9}$$

$$f_\mu = \exp \left[\frac{-3.4}{\left(1 + Re_T / 50 \right)^2} \right], \tag{10}$$

$$Re_T = \frac{k^2}{\nu \varepsilon}, \tag{11}$$

$$f_2 = 1 - 0.3 \exp \left(-Re_T^2 \right). \tag{12}$$

The models parameters are $\sigma_k = 1$, $\sigma_\varepsilon = 1.3$, $C_{\varepsilon 1} = 1.44$, $C_{\varepsilon 2} = 1.92$ and $C_\mu = 0.09$. The boundary conditions at a solid wall are homogeneous Dirichlet for both k and ε. Note that the energy dissipation rate ε used in the Launder-Sharma model is a modification of the true dissipation rate $\tilde{\varepsilon} = \varepsilon + 2\nu(\partial\sqrt{k}/\partial y)^2$. The last terms on the right hand side of (5)–(6) are in the original Launder-Sharma model expressed as

$$2\nu \left(\frac{\partial \sqrt{k}}{\partial y} \right)^2, \quad 2\nu\nu_T \left(\frac{\partial^2 U}{\partial y^2} \right),$$

where y is the wall normal direction and U is the velocity tangential to the wall. To overcome this coordinate dependency the terms have been implemented as seen in the last terms of (5) and (6). The justification for this implementation is that the terms are only significant near a wall, where the gradient in the wall normal direction will be totally dominating. The implemented coordinate independent terms will thus approach the true model in regions where it is significant, regardless of geometry and orientation of the wall.

Since the focus of this paper is the finite element formulation of a RANS model and its implementation in the FEniCS environment, the specific turbulence model should be numerically representative. The present k-ε model is highly nonlinear and represents in a way a "worst case" with demands to nonlinear iteration strategies. On the contrary, the model has simple boundary conditions, which makes our first exploration of finite elements and FEniCS simpler.

Case study

Fully developed plane channel flow

The fully developed plane channel flow is a simple, statistically one-dimensional, flow that is used as a first test case for most turbulence models. The flow is set in between two flat plates separated by a distance $2H$ in the y-direction, and is driven by a constant negative pressure gradient in the streamwise x-direction. The flow in the streamwise direction is periodic, and the (z)-direction is homogeneous. The channel in this work is operated at $Re_\tau = 395$, where

$$\mathrm{Re}_\tau = \frac{u_\tau H}{\nu}. \tag{13}$$

The friction velocity $u_\tau = \sqrt{\nu \partial U/\partial y}$, where y is the coordinate direction normal to the wall. The constant negative pressure gradient is determined by u_τ as

$$-\frac{\partial p}{\partial x} = \frac{u_\tau^2}{H} \tag{14}$$

All quantities $\boldsymbol{U}$, k and ε have homogeneous Dirichlet boundary conditions on the walls. Note that since the pressure gradient in the streamwise direction is constant, it is not necessary to solve for P in the channel, but we want to apply a general 2D/3D solver which will have the pressure as unknown also in the straight channel case. The applied boundary conditions are summarized in Table 1.

Variable	Inflow / Outflow	Wall
k	Periodic	0
ε	Periodic	0
u	Periodic	0
p	Dirichlet	$\frac{\partial p}{\partial n} = 0$

Table 1: Summary of boundary conditions

Periodic or cyclic boundary conditions means that for each iteration, the value of a variable at a specific node is mapped from the outflow boundary to the corresponding inflow boundary node.

Numerical Methods

The Reynolds Averaged Navier-Stokes Equations

To solve the transient Reynolds averaged Navier-Stokes equations (3) and (4), we adapt the ideas of the original splitting scheme by Chorin and Temam [8]. Steady-state problems must be solved as a transient development of the flow towards the steady state.

We have an explicit handeling of the convection, and treat diffusion implicitly in (3)–(4). It is well known that this requires implicit handling of the pressure and the $\boldsymbol{U}$ in the continuity equation. The weak formulation of the resulting time-discrete system then becomes: find $(\boldsymbol{U}^{l+1}, P^{l+1}) \in V_u \times V_p$, where $V_u = \left[H^1(\Omega)\right]^d, V_p = L^2(\Omega)$ such that

$$\begin{aligned} &\left(\boldsymbol{U}^{l+1}, \boldsymbol{v}\right) + \left(\left[\nu + \nu_T^l\right]\boldsymbol{\nabla}\boldsymbol{U}^{l+1}, \boldsymbol{\nabla}\boldsymbol{v}\right) = \\ &-\left(P^{l+1}, \boldsymbol{\nabla}\cdot\boldsymbol{v}\right) + \left(\boldsymbol{U}^l, \boldsymbol{v}\right) - \Delta t\left[\left(\boldsymbol{U}^l\cdot\boldsymbol{\nabla}\boldsymbol{U}^l, \boldsymbol{v}\right) + \left(\boldsymbol{f}^l, \boldsymbol{v}\right)\right] \quad \forall \boldsymbol{v} \in V_u \end{aligned} \tag{15}$$

$$\left(\boldsymbol{\nabla}\cdot\boldsymbol{U}^{l+1}, q\right) = 0 \quad \forall q \in V_p \tag{16}$$

To solve this implicit system in $\boldsymbol{U}^{l+1}$ and P^{l+1}, we employ the common ideas of first computing a tentative velocity, and then constrain the pressure such that $\boldsymbol{U}^{l+1}$ fulfills the continuity equation. The resulting four updating equations of the Chorin and Temam scheme can be written as

$$\begin{aligned}&(\boldsymbol{U}^*, \boldsymbol{v}) + \left(\left[\nu + \nu_T^l\right]\boldsymbol{\nabla}\boldsymbol{U}^*, \boldsymbol{\nabla}\boldsymbol{v}\right) = \\ &-\beta\left(P^l, \boldsymbol{\nabla}\cdot\boldsymbol{v}\right) + \left(\boldsymbol{U}^l, \boldsymbol{v}\right) - \Delta t\left[(\boldsymbol{U}^l\cdot\boldsymbol{\nabla}\boldsymbol{U}^l, \boldsymbol{v}) + (\boldsymbol{f}^l, \boldsymbol{v})\right] \quad \forall \boldsymbol{v}\in V_u\end{aligned} \tag{17}$$

$$(\boldsymbol{\nabla}\phi, \boldsymbol{\nabla}q) = (-\frac{1}{\Delta t}\boldsymbol{\nabla}\cdot\boldsymbol{U}^*, q) \quad \forall q\in V_q \tag{18}$$

$$(\frac{1}{\rho}P^{l+1}, q) = (\frac{1}{\rho}P^l + \phi, q) \quad \forall q\in V_q \tag{19}$$

$$(\boldsymbol{U}^{l+1}, \boldsymbol{v}) = (\boldsymbol{U}^* - \Delta t\boldsymbol{\nabla}\phi, \boldsymbol{v}) \quad \forall \boldsymbol{v}\in V_v \tag{20}$$

In the original formulation, the parameter β was zero, but may be (e.g.) chosen as unity to incline the pressure at the previous time step. Further, the variable ϕ denotes the pressure difference from a time step to another, $\phi = p^{l+1} - p^l$, and is calculated as shown above. Finally, the pressure and velocity is updated, before one proceeds to the next time level.

The k-ε Equations

The weak form of the k-ε equations reads: find $(k^{l+1}, \varepsilon^{l+1}) \in V_v \times V_q$ where $V_v, V_q = H^1(\Omega)$ such that

$$\begin{aligned}&\left(\dot{k}, v\right) + (\boldsymbol{U}\cdot\boldsymbol{\nabla}k, v) = \left(\left[\nu + \frac{\nu_T}{\sigma_k}\right]\boldsymbol{\nabla}k, \boldsymbol{\nabla}v\right) + (P_k, v) - \\ &\qquad (\varepsilon, v) + \left(2\nu\left(\left\|\boldsymbol{\nabla}\sqrt{k}\right\|\right)^2, v\right), \quad \forall v\in V_v\end{aligned} \tag{21}$$

$$\begin{aligned}&(\dot{\varepsilon}, q) + (\boldsymbol{U}\cdot\boldsymbol{\nabla}\varepsilon, q) = \left(\left[\nu + \frac{\nu_T}{\sigma_\varepsilon}\right]\boldsymbol{\nabla}\varepsilon, \boldsymbol{\nabla}q\right) + \left(\left(C_{\varepsilon 1}P_k - f_2 C_{\varepsilon 2}\varepsilon\right)\frac{\varepsilon}{k}, q\right) + \\ &\qquad \left(2\nu\nu_T\left\|\boldsymbol{\nabla}^2 u\right\|^2, q\right), \quad \forall q\in V_q\end{aligned} \tag{22}$$

Here, the dot notation denotes a partial derivative in time, e.g., $\dot{k} = \partial k/\partial t$.

It remains to construct a time discretization scheme for the k and ε equations. Our choice is a semi-implicit Crank-Nicolson scheme as described in [9]. The time-centered Crank-Nicolson scheme gives nonlinear equations to be solved at each time level. We therefore introduce the index n for counting nonlinear iterations such that $k^{l+1,n}$ means k in iteration n at time level $l+1$. The notation k^l implies a converged value of k at the previous time level. It is often necessary to apply under relaxation to achieve convergence. In an iteration, we therefore seek a solution $k^{l+1,*}$ of the discrete, linearized partial differential equation and then set the iteration value in iteration n as

$$k^{l+1,n} = \omega_k k^{l+1,*} + (1-\omega_k)k^{l+1,n-1}, \tag{23}$$

where ω_k is a relaxation parameter for k. The start guess $k^{l+1,0}$ for the nonlinear iterations at a new time level is naturally taken as the solution at the previous time level: $k^{l+1,0} = k^l$. The notation and ideas introduced here for k are applied to ε and other quantities of interest as well.

At each time level, the velocity and pressure are first updated, and then we enter a loop to solve the k and ε equations. In each pass of this loop, the k equation is solved and k is relaxed. Then the ε equation is solved and ε is relaxed. Finally, ν_T is also relaxed (usually requiring a relaxation parameter different from those used for k and ε). The equations to be solved for a new k and ε at a new iteration n at the new time level $l+1$ can then be written as:

$$\begin{aligned}
&\left(\frac{k^{l+1,*}-k^{l}}{\Delta t},v\right)+\frac{1}{2}\left(\boldsymbol{U}^{l+1}\cdot\boldsymbol{\nabla}k^{l+1,*},v\right)+\frac{1}{2}\left(\boldsymbol{U}^{l}\cdot\boldsymbol{\nabla}k^{l,n-1},v\right)=\\
&\frac{1}{2}\left(\left[\nu+\frac{\nu_T^{l,n-1}}{\sigma_k}\right]\boldsymbol{\nabla}k^{l+1,*},\boldsymbol{\nabla}v\right)+\frac{1}{2}\left(\left[\nu+\frac{\nu_T^{l,n-1}}{\sigma_k}\right]\boldsymbol{\nabla}k^{l,n-1},\boldsymbol{\nabla}v\right)+\\
&\left(P_k^{l+1},v\right)-\left(\varepsilon^{l,n-1}\frac{k^{l+1,*}}{k^{l,n-1}},v\right)+\left(2\nu\left(\left\|\boldsymbol{\nabla}\sqrt{k^{l,n-1}}\right\|\right)^2,v\right)
\end{aligned}\tag{24}$$

$$\begin{aligned}
&\left(\frac{\varepsilon^{l+1,*}-\varepsilon^{l}}{\Delta t},q\right)+\frac{1}{2}\left(\boldsymbol{U}^{l+1}\cdot\boldsymbol{\nabla}\varepsilon^{l+1,*},q\right)+\frac{1}{2}\left(\boldsymbol{U}^{l}\cdot\boldsymbol{\nabla}\varepsilon^{l},q\right)=\\
&\frac{1}{2}\left(\left[\nu+\frac{\nu_T^{l+1,n-1}}{\sigma_\varepsilon}\right]\boldsymbol{\nabla}\varepsilon^{l+1,*},\boldsymbol{\nabla}q\right)+\frac{1}{2}\left(\left[\nu+\frac{\nu_T^{l}}{\sigma_\varepsilon}\right]\boldsymbol{\nabla}\varepsilon^{l},\boldsymbol{\nabla}q\right)+\\
&\left(\left(C_{\varepsilon 1}P_k^{l+1}-f_2^{l+1,n-1}C_{\varepsilon 2}\varepsilon^{l+1,*}\right)\frac{\varepsilon^{l+1,n-1}}{k^{l+1,n}},q\right)+\left(2\nu\nu_T^{l+1,n-1}\left\|\boldsymbol{\nabla}^2\boldsymbol{U}^{l+1}\right\|^2,q\right)
\end{aligned}\tag{25}$$

The source terms have been treated either as fully implicit or explicit. In turbulence modelling it is common to compute production terms explicitly and dissipation terms implicitly. The reason is simply that the production terms, P_k, are positive and if incorporated in the mass matrix, they would decrease the tridiagonal dominance, which destabilizes the iterative conjugate gradient linear algebra solvers. The dissipation terms, on the contrary, are negative and will increase the diagonal dominance, thereby stabilizing the solvers. The same treatment is used in OpenFOAM.

The damping functions f_2 and f_μ are evaluated such that:

$$f_2^{l+1,n}=f(k^{l+1,n},\varepsilon^{l+1,n-1}),\text{ and }f_\mu^{l+1,n}=f(k^{l+1,n},\varepsilon^{l+1,n})\,.$$

The turbulent viscosity is computed as

$$\nu_T^{l+1,*}=C_\mu f_\mu^{l+1,n}\frac{(k^{l+1,n})^2}{\varepsilon^{l+1,n}}\tag{26}$$

and then relaxed:

$$\nu_T^{l+1,n}=\omega_T\nu_T^{l+1,*}+(1-\omega_T)\nu_T^{l+1,n-1}\,.\tag{27}$$

To avoid very large values of ν_T during iterations, it is common to bound the value as follows:

$$\nu_T=\nu 10^s\text{ if }\nu_T>\nu 10^s\tag{28}$$

for s equal to a large number, say 10^3. This adjustment of ν_T is applied to all grid points where ν_T (and k and ε) are defined. Since large ν_T values are normally caused by (too) small ε values, we also adjust ε if ν_T is adjusted:

$$\varepsilon=C_\mu f_\mu k^2/\nu_T\,.\tag{29}$$

The last term of (25) cannot, in standard finite element methods, be computed as it stands. Our treatment of this term consists in introducing a new auxiliary variable $\boldsymbol{L}$ for $\boldsymbol{\nabla}^2\boldsymbol{U}$. This $\boldsymbol{L}$ is computed by a finite element formulation of

$$\boldsymbol{L} = \boldsymbol{\nabla}^2\boldsymbol{U},$$

which reads: find $\boldsymbol{L} \in V_v$ such that

$$(\boldsymbol{L}^{l+1}, \boldsymbol{v}) = -(\boldsymbol{\nabla}\boldsymbol{U}^{l+1}, \boldsymbol{\nabla}\boldsymbol{v}) + \text{boundary terms} \quad \forall \boldsymbol{v} \in V_v \tag{30}$$

The last term of (25) can then be calculated as $\left(2\nu\nu_T^{l+1,n-1}\left\|\boldsymbol{L}^{l+1}\right\|^2, q\right)$, if $\boldsymbol{L}^{l+1}$ is computed right after $\boldsymbol{U}^{l+1}$ is available, and before k^{l+1} and ε^{l+1} are to be found.

Implementation in FEniCS

We can now summarize the solution algorithm for our k-ε model:

$while\ t <= T$
  solve the averaged Navier-Stokes eqs. (17) for $\boldsymbol{U}^{l+1}$ and P^{l+1}
  solve (30) for $\boldsymbol{L}^{l+1}$
  while not converged and $n < n_{\max}$:
    solve (24) for $k^{l+1,*}$
    relax $k^{l+1,*}$ to $k^{l+1,n}$
    solve (25) for $\varepsilon^{l+1,*}$
    relax $\varepsilon^{l+1,*}$ to $\varepsilon^{l+1,n}$
    compute ν_T from (26)–(26)

There are many ways to adjust the algorithm if one encounteres convergence problems. For example, the update of $\boldsymbol{U}$ and P can be placed inside the nonlinear iteration loop. It is also possible to merge the k and ε equations into one, simultaneous matrix system, but this latter approach will often lead to inferior convergence compared to a segregated approach, as in the algorithm above, for the present k-ε model.

The code for defining the various terms in the various equations and solving for the unknown is very similar, so here we limit the attention to how the ε equation is implemented.

There is a very close relationship between a variational formulation like (25) and the corresponding FEniCS code. For example, a term $(\boldsymbol{\nabla}\varepsilon, \boldsymbol{\nabla}q) = \int_\Omega \boldsymbol{\nabla}\varepsilon \cdot \boldsymbol{\nabla}q d\Omega$ is typically implemented as `dot(grad(eps), grad(q))*dx` in Python. The `eps` and `q` must be defined as trial and test functions first, over some mesh consisting of the specified type of elements. Figure 1 shows the code for generating the finite element space V_q and the corresponding functions. We use the Continuous Galerkin method, known as the name `CG` in the code, and first order Lagrange (triangular) elements (specified by the last argument `1`). The `mesh` object holds a finite element mesh read from file or generated in some fashion.

We can now start defining the various terms in (25). Logical variable names are introduced: `e_term_i` means a variable `e` (for ε), a term `term` (`transient`, `convective`, `diffusion`, `production`, etc.), and implicit (`i`) handling of the term, implying that the term will enter the coefficient matrix in the linear system to be solved at the current iteration level (n). A postfix `e`,

```
V_q =  FunctionSpace(mesh, "CG", 1)
q  = TestFunction (V_q)
e  = TrialFunction(V_q)
```

Figure 1: Creation of finite element spaces and their functions

as in `e_term_e`, indicates that the term is treated explicitly, i.e., it enters the right-hand side of the linear system in this iteration. The variable `e` holds the most recent value in the nonlinear iteration at the current time step, while `e0` is the value at the previous time step, i.e., ε^l. A similar notation is used for other quantities. Figure 2 shows how some typical terms in the ε equation are defined. Notice the simplicity in the specification of the variational forms – the program is very close to the mathematical notation of a finite element problem.

```
e_transient_i   = e*q
e_transient_e   = e0*q

e_convective_i  = dot(U, dot(grad(e),  q))
e_convective_e  = dot(U, dot(grad(e0), q))

e_diffusion_i   = (nu+nu_T0*/ce)*dot(grad(e),  grad(q))
e_diffusion_e   = (nu+nu_T0*/ce)*dot(grad(e0), grad(q))

e_dissipation_i = f2*ce2*e*(e0/k0)*q
e_production_e  = ce1*P*(e0/k0)*q
L_e    = 2*nu*nu_T0*L*L*q
```

Figure 2: Definition of weak forms in the ε-equation

We are now in a position to form a variational problem $a_\varepsilon(\varepsilon, q) = L_\varepsilon(q)$ for the ε equation, i.e., the bilinear form a_ε holds all the terms that are treated implicitly and that will enter the coefficient matrix and the unknown vector of new ε values, while the linear form L_ε holds all known (explicit) terms that enter the right-hand side of the linear system. Figure 3 displays the specifications of the forms.

```
a_e = e_transient_i - 0.5*dt*(- e_convective_i - e_diffusion_i \
                    - 2.0*e_dissipation_i)*dx

L_e = e_transient_e + 0.5*dt*(- e_convective_e - e_diffusion_e \
                    + 2.0*e_production_e + 2.0*e_wall_fcn_e )*dx
```

Figure 3: FEniCS implementation of the final bilinear and linear forms

Having defined the form $a_\varepsilon(\varepsilon, q) = L(q)$ in a finite element problem, we can solve the equation system implied by this form by a single statement as shown in Figure 4, where `e1` in the program corresponds to $\varepsilon^{l+1,*}$. Instead of solving the variational form, we can assemble a coefficient matrix A from $a_\varepsilon(\varepsilon, q)$ and a right-hand side b from $L_\varepsilon(q)$. Then we have full control what we do with A and b when solving the linear system $A\varepsilon = b$. For example, we may combine A and b with coefficient matrices and right-hand sides from other equations, into a block matrix

system to construct implicit methods, etc. There is a great flexibility in working both with finite element forms and/or the associated matrices and vectors. The solver for the linear system can be specified in detail, and the current choice is uBLAS, [5]. Figure 5 shows a part of the implementation. Notice that equation 30 is only dependent of U, and placed outside the iteration loop.

```
# bc_e holds the essential (Dirichlet) boundary conditions
e_problem = VariationalProblem(a_e, L_e, bc_e)
e1 = e_problem.solve()
```

Figure 4: Solving the variational formulation

Figure 5 shows the main program, i.e., the overall solution algorithm as presented earlier.

```
while t <= T:
    U_problem = VariationalProblem(a_ns, L_ns, bc_ns)
    U1 = U_problem.solve()
    L_problem = VariationalProblem(a_l, L_l, bc_l)
    L = L_problem.solve()

    while not converged and n < n_max:
        k_problem = VariationalProblem(a_k, L_k, bc_k)
        k1 = e_problem.solve()
        k = omega_k*k1 + (1-omega_k)*k0
        e_problem = VariationalProblem(a_e, L_e, bc_e)
        e1 = e_problem.solve()
        e = omega_e*e1 + (1-omega_e)*e0
        eval_nu_T()
```

Figure 5: Overall solution algorithm

OpenFOAM implementation

OpenFOAM (Open Source Field Operation and Manipulation) is free Computational Fluid Dynamics (CFD) software designed for the solution of problems in continuum mechanics. OpenFOAM is first and foremost a C++ library that provide numerical schemes implemented in the traditional finite volume framework, with solvers that are known to be efficient for continuum mechanics problems. In OpenFOAM new problems are solved by first creating meshes on a given format and then by pulling models, numerical schemes and solvers from the vast C++ library.

The Launder & Sharma turbulence model is part of the OpenFOAM library of incompressible Reynolds Averaged turbulence models. In this paper we use this model with the most standard low order finite volume numerical schemes, that is linear interpolation for face values and central differencing for face gradients. For the channel the steady-state boundaryFoam solver (also part of the main OpenFOAM distribution) is used. This solver is designed for statistically one-dimensional boundary layer problems, which is appropriate in the present channel flow case. However, OpenFOAM always operates with three spacial dimensions. The one-dimensionality of the problem is thus obtained by creating a three-dimensional mesh with merely one control

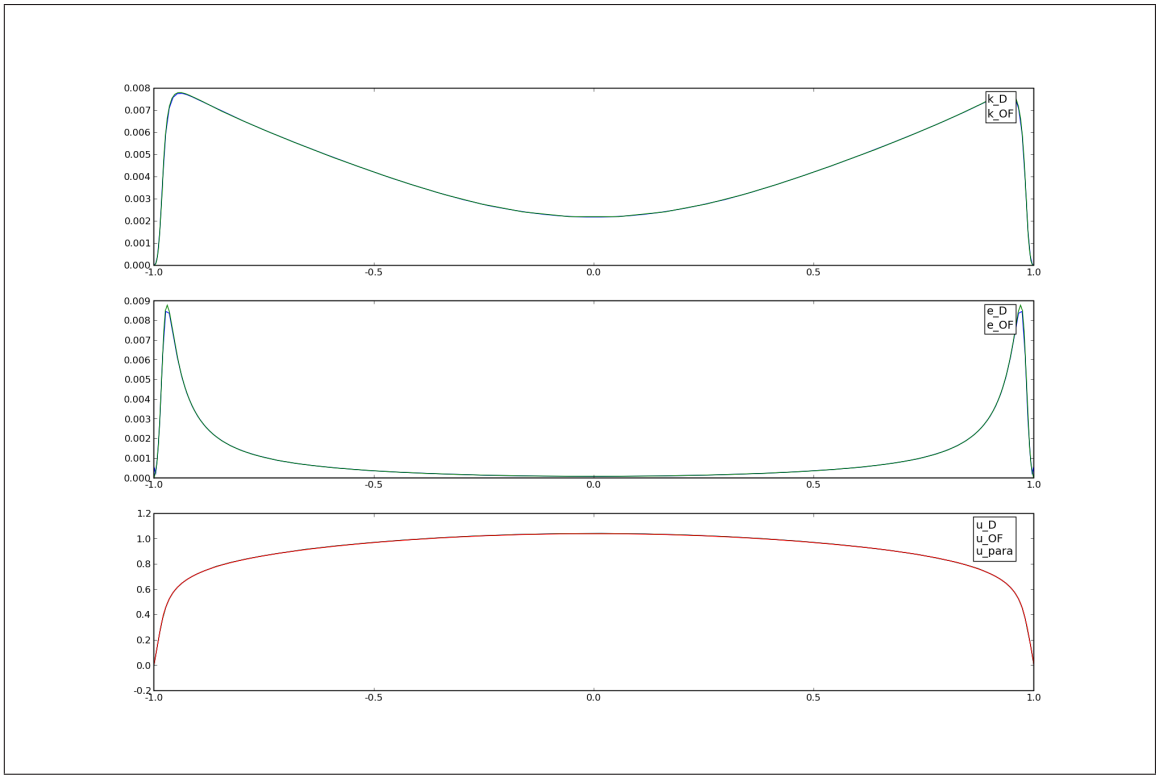

Figure 6: k-ε implementation in FEniCS versus OpenFOAM

volume in the streamwise and spanwise directions and by imposing periodic (termed cyclic in OpenFOAM) and symmetry (termed empty in OpenFOAM) boundary conditions respectively. This means that the solver incorporates convection terms even though they are all identical to zero. The steady-state solver updates the U, k and ε parameters in a segregated, under-relaxed fashion and with additional under-relaxation on ν_T for stability. The initial guessed solution is uniform, with values $U = 0.914$, $k = 0.01$ and $\varepsilon = 0.01$. Underrelaxation factors are set to 0.7, except for U that uses 0.5.

Numerical Verification

We have used the FEniCS implementation to simulate turbulent flow in a channel, using a two-dimensional domain $\Omega = \big[0.0, 0.1\big] \times \big[0.0, 2.0\big]$, with a mesh size of 3 nodes in the x-direction and 100 nodes in the y direction. The mesh spacing in the y-direction is determined by a growth factor of 1.1, from the wall to the center of the channel. The time step is naturally chosen according to the CFL condition. However, we have chosen to perform only a single iteration at each time level and instead reduce the time step by a factor of 10. The initial conditions have been set to constant values for all unknown quantities over the entire domain, i.e., $k = 0.01$ and $\varepsilon = 0.01$, while the velocity and pressure starts from zero.

Figure 6 shows the solutions of k-ε and U from the top. The agreement is apparently good, but the log vs. log plot in Figure 7 revels differences with the openFOAM results.

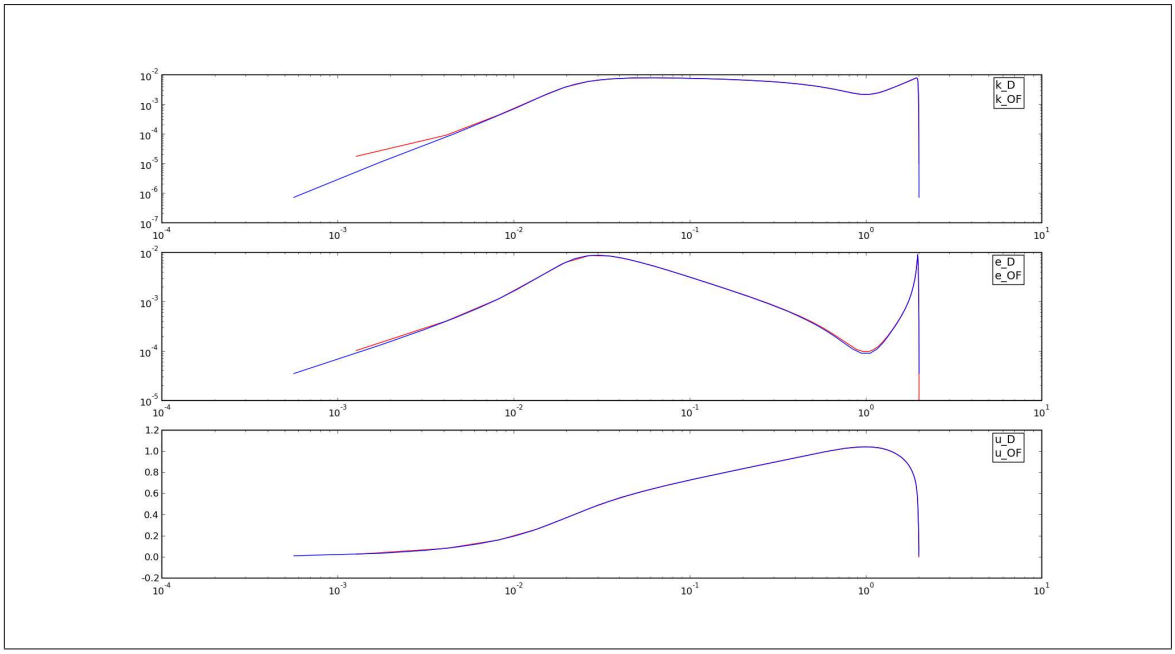

Figure 7: k-ε implementation in FEniCS versus OpenFOAM, logplot.

Discussion

There is a great need for algorithmic flexibility when new turbulence models are to be implemented and tested, primarily because the PDEs are highly nonlinear and require iteration strategies tailored to the problem at hand. This algorithmic flexibility is best obtained by allowing the computational turbulence researcher to create computer code for the equations and the overall solution algorithm. Such code is normally very complicated in a standard programming language as Fortran, and programming is consequently non-trivial and often error-prone. Using high-level languages, such as Python or C++, makes it possible to design appropriate abstractions so that the differences between a mathematical formulation of the solution strategy and the actual code are small. FEniCS is a recent programming environment for this purpose, and the goal of this paper has been to explore the possibilities for using FEniCS in computational turbulence. The resulting program is short and lend itself easily to tailored adjustments of iteration strategies, implicit/explicit handling of terms, higher-order approximations, other splittings of the PDE system, and so forth.

We experienced no technical difficulties in the current finite element implementation of the Launder-Sharma k-ε model. Therefore, we expect the finite element method to have a significant potential also for computational turbulence, despite being used to a very little extent. The wall function for the ε-equation, described by equation 30, needed special treatment in the finite element framework and a demand for solving for an auxiliary field quantity. This special treatment is not necessary in the finite volume method and may be seen as an extra complication when finite elements are used to solve the current system of PDEs.

The current k-ε model employs simple boundary conditions for k and ε. Other RANS models may involve implicit boundary conditions, i.e., equations containing both k and ε. This extra difficulty needs to be resolved in a finite element (and FEniCS) context and will be a topic of future work.

The FEniCS implementation is not optimized with respect to the channel problem we as ver-

ification case in this paper. Instead, the implementation is transient and generic, and may be used to handle any domain in two or three space dimensions. The openFOAM implementation we compare with is a stationary, one-dimensional solver tailored to steady-state channel flow. Hence, it does not make sense to compare CPU time or the behavior of nonlinear iterations in the openFOAM solver versus the slow time evolution used in the FEniCS solver.

References

[1] Featflow, http://www.featflow.de.

[2] The fenics project (http://www.fenics.org).

[3] http://www.cfd-online.com/links/soft.html.

[4] Openfoam ®: The open source cfd toolbox (http://www.opencfd.co.uk/openfoam/).

[5] ublas c++ template class library.

[6] W. P.Jones and B. E.Launder The prediction of laminarization with a two-equation model of turbulence *Int. J. Heat Mass Transfer*, **vol.15**, 301–314, 1972.

[7] B. E.Launder and B. I.Sharma Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc *Letters in Heat and Mass Transfer*, **vol.1**(2), 131–138, 1974.

[8] R.Temam Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires *Arc. Ration. Mech. Anal.*, **vol.32**, 377–385, 1969.

[9] H. K.Versteeg and W.Malalasekera *An Introduction to Computational Fluid Dynamics: The Finite Volume Method* Prentice Hall, 2007.