explains a collection of PDE solvers in detail: the Poisson equation, the mixed formulation for the Poisson equation, the Biharmonic equation, the equations of hyperelasticity, the Cahn-Hilliard equation, and the incompressible Navier–Stokes equations. Both Python and C++ versions of these solvers are explained. An eigenvalue solver is also documented. In the `dolfin/demo` directory of the DOLFIN source code tree you can find programs for these and many other examples, including the advection-diffusion equation, the equations of elastodynamics, a reaction-diffusion equation, various finite element methods for the Stokes problem, discontinuous Galerkin methods for the Poisson and advection-diffusion equations, and an eigenvalue problem arising from electromagnetic waveguide problem with Nédélec elements. There are also numerous demos on how to apply various functionality in FEniCS, e.g., mesh refinement and error control, moving meshes (for ALE methods), computing functionals over subsets of the mesh (such as lift and drag on bodies in flow), and creating separate subdomain meshes from a parent mesh.

The project CBC.Solve (`https://launchpad.net/cbc.solve`) offers more complete PDE solvers for the Navier–Stokes equations (Chapter 24), the equations of hyperelasticity (Chapter 29), fluid–structure interaction (Chapter 24), viscous mantle flow (Chapter 33), and the bidomain model of electrophysiology. Another project, CBC.RANS (`https://launchpad.net/cbc.rans`), offers an environment for very flexible and easy implementation of systems of nonlinear PDEs in general, and a collection of Navier–Stokes solvers and turbulence models in particular Mortensen et al. [2011b,a]. For example, CBC.RANS contains an elliptic relaxation model for turbulent flow involving 18 nonlinear PDEs. FEniCS proved to be an ideal environment for implementing such complicated PDE models.

1.8 Miscellaneous topics

1.8.1 Glossary

Below we explain some key terms used in this tutorial.

FEniCS: name of a software suite composed of many individual software components (see `fenicsproject.org`). Some components are DOLFIN and Viper, explicitly referred to in this tutorial. Others are FFC and FIAT, heavily used by the programs appearing in this tutorial, but never explicitly used from the programs.

DOLFIN: a FEniCS component, more precisely a C++ library, with a Python interface, for performing important actions in finite element programs. DOLFIN makes use of many other FEniCS components and many external software packages.

Viper: a FEniCS component for quick visualization of finite element meshes and solutions.

UFL: a FEniCS component implementing the unified form language for specifying finite element forms in FEniCS programs. The definition of the forms, typically called `a` and `L` in this tutorial, must have legal UFL syntax. The same applies to the definition of functionals (see Section 1.1.7).

Class (Python): a programming construction for creating objects containing a set of variables and functions. Most types of FEniCS objects are defined through the class concept.

Instance (Python): an object of a particular type, where the type is implemented as a class. For instance, `mesh = UnitInterval(10)` creates an instance of class `UnitInterval`, which is reached by the name `mesh`. (Class `UnitInterval` is actually just an interface to a corresponding C++ class in the DOLFIN C++ library.)

Class method (Python): a function in a class, reached by dot notation: `instance_name.method_name`.

`self` parameter (Python): required first parameter in class methods, representing a particular object of the class. Used in method definitions, but never in calls to a method. For example, if `method(self,`