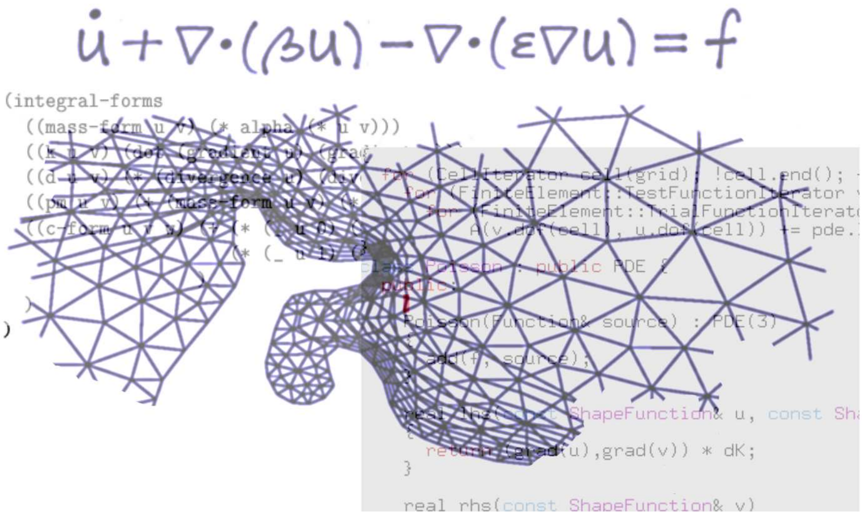
CHALMERS

FINITE ELEMENT CENTER



PREPRINT 2003–21

The FEniCS project

T. Dupont, J. Hoffman, C. Johnson, R. Kirby, M. Larson, A. Logg, and R. Scott

Chalmers Finite Element Center
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg Sweden 2003

CHALMERS FINITE ELEMENT CENTER

Preprint 2003–21

The FEniCS project

T. Dupont, J. Hoffman, C. Johnson, R. Kirby, M. Larson,
A. Logg, and R. Scott

CHALMERS

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg Sweden
Göteborg, November 2003

The FEniCS project
T. Dupont, J. Hoffman, C. Johnson, R. Kirby, M. Larson,
A. Logg, and R. Scott
NO 2003–21
ISSN 1404–4382

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone: +46 (0)31 772 1000
Fax: +46 (0)31 772 3595
www.phi.chalmers.se

Printed in Sweden
Chalmers University of Technology
Göteborg, Sweden 2003

THE FENICS PROJECT

T. DUPONT, J. HOFFMAN, C. JOHNSON, R. KIRBY, M. LARSON, A. LOGG, AND R. SCOTT

Abstract. The goal of the FEniCS project is to develop open-source software for the automation of Computational Mathematical Modeling (CMM). FEniCS is a joint project between the Toyota Technological Institute at Chicago, the University of Chicago, and Chalmers University of Technology.

The vision of FEniCS is to set a new standard in CMM, which can be described as the Automation of CMM, towards the goals of generality, efficiency, and simplicity, concerning mathematical methodology, implementation, and application.

1. Modern Society and CMM

The Industrial Society is characterized by automated production of material goods and the Information Society is characterized by automated production of virtual goods. Automation has opened for mass production of both material goods and information, and may thus be viewed as the basis of our Modern Society, with mass consumption of both material and virtual goods.

The Industrial Society developed along with the development of Calculus of differential/integral equations and Science, starting in the 18th century, and the Information Society based on the computer has developed along with mathematical logic, discrete and numerical mathematics.

Mathematics thus plays an increasingly important role in the development of Modern Society, as a language for Science and a theoretical and practical tool for Computation. *Computational Mathematical Modeling* (CMM) may be viewed as the modern expression of the *Basic Principle of Science*: formulating equations (modeling) and solving equations (computation).

Automation plays a key role in modern society, e.g., to improve quality of life and to reduce environmental impact. This includes mass production of both material goods, such as food, clothes and cars, and virtual goods in the form of digitized word, image and

---

Date: November 6, 2003.

Key words and phrases. Automation, computational mathematical modeling, variational form, finite element method, Galerkin, adaptivity, open-source software, dolfin, analysa, fiat.

Todd Dupont, dupont@cs.uchicago.edu, Dept. of Computer Science, University of Chicago, Ill USA,
Johan Hoffman, hoffman@courant.nyu.edu, Courant Inst. of Mathematical Sciences, New York, USA
Claes Johnson, claes@math.chalmers.se, Dept. of Computational Mathematics, Chalmers, Sweden
Robert C. Kirby, kirby@cs.uchicago.edu, Dept. of Computer Science, University of Chicago, Ill USA
Mats Larson, mgl@math.chalmers.se, Dept. of Computational Mathematics, Chalmers, Sweden
Anders Logg, logg@math.chalmers.se, Dept. of Computational Mathematics, Chalmers, Sweden
Ridgway Scott, ridg@cs.uchicago.edu, Dept. of Computer Science, University of Chicago, Ill USA.

sound. Computational simulation is often a key step in the further refinement of designs of new products or manufacturing processes. It plays an important role in producing both physical and informational goods.

The term CMM is used to describe precisely the field of simulation based on mathematical models using rigorous computational mathematics. Thus, the word 'mathematical' modifies both 'computational' as well as 'modeling'. Standard models involve combinations of an already broad range of forms, including algebraic, differential and integral equations. However, increasingly more complex models are also appearing. This makes it imperative that general computational modeling be implemented in flexible, extensible (FE) systems.

1.1. **CMM and Mathematics.** CMM may be viewed as *Constructive Mathematics* and is thus a fundamental part of Mathematics. With the development of the computer, the world of Constructive Mathematics is now quickly expanding because more mathematical objects may be constructed and processed.

It is obvious that also other parts of mathematics play a key role in CMM. One example of this is the remarkable development of the field of nonlinear partial differential equations over the last half century. We are now much better able to understand whether a novel model of natural phenomena is sensible or not. This is a key step in the process of evaluating a novel mathematical model being proposed for new phenomena. It should therefore be possible to write novel equations in a simple language and perform numerical simulations reliably and automatically.

The increased use of CMM in the field of partial differential equations is now contributing to the further development of the field. For example, the Clay Prize problem on existence of smooth solutions of the Navier–Stokes equations, still unsolved 70 years after Leray's first attempt, is now being attacked in part by computational methods.

Another significant development in mathematics is the theory of finite elements (another FE) in the mathematical community over the last three decades. However, these developments have not been fully reflected in general software. Thus, an initial step will be to automate the implementation of the mathematical theory of the finite element method. This will provide a basis for development of other technologies in our general system.

The finite element method is the Galerkin method with a particular class of approximating spaces. The Galerkin method with general approximation spaces would be a logical next step, including spectral methods, wavelets, variational difference methods, and methods for Fourier integral operators.

1.2. **CMM in Science and Engineering.** CMM plays an increasingly important role in Science and Engineering as the capability of computational solution of (differential/integral) equations is quickly expanding. Computational simulation of real world phenomena from quantum physics and chemistry through molecular biology to geophysics and astronomy, is becoming possible for the first time, and entirely new perspectives and means for progress are opening.

One description of the scientific method would be the discovery and exploitation of mathematical equations to describe natural phenomena, or shortly: formulating equations (modeling) and solving equations (computation). A key aspect of modern computational explorations of science and engineering questions is the need for varying the physical models. This is sometimes referred to as multi-physics simulation. But we emphasize that modern computational science (also known as CS) often is exploring the space of new models, not just working with a combination of fixed models.

The use of computational modeling is constantly growing as new fields start to use CMM, and fields which have long used CMM explore new types of phenomena. This clearly requires a new level of automation in CMM. We will argue that it is even possible to apply ideas from learning theory to the adaptive discovery of models, using CMM recursively.

1.3. **CMM and Computer Science.** Computer Science (CS) has been described as a study of automation. Alternatively, CS may be viewed as a development of Constructive Mathematics. The original motivation to develop computers was to replace the drudgery and errors of human groups using desk calculators in basics tasks of CMM, such as solving differential equations. In the early development of CS as a discipline, computational mathematics thus played an important role, but gradually the focus in CS shifted to other issues such as AI, programming languages, data bases, compilers, etc.

Today, there are many aspects of CMM that can benefit from recent developments in CS. Conversely, aspects of CMM related to adaptivity pose challenges to CS connecting to modern programming tools and machine learning. Given the role of CMM in the automation process, it is clearly a substantial part of CS. Moreover, there is a two-way relationship in that automation itself also plays a role in the process of computational modeling. Thus, there is a natural recursive relationship between CMM, automation and thus CS.

Specific aspects of CMM that can benefit from recent developments in CS include the following. Algorithms in computational geometry play an obvious role, but also extensible systems can benefit from developments in languages and compilers. There are many aspects of CMM that can be viewed from the perspective of machine learning. Data mining and data base issues are also increasingly relevant to CMM. Correspondingly, CMM places novel emphasis on certain issues in CS. Performance (of floating-point computation) is critical in CMM, since it is characterized as numerically intensive (NI).

2. The vision of FEniCS

The vision of **FEniCS** is to set a new standard in CMM, which can be described as the *Automation of CMM*, towards the goals of *generality*, *efficiency*, and *simplicity*, concerning *mathematical methodology*, *implementation*, and *application*.

The goals of generality and simplicity reflect Leibniz' principle of the "Best Possible of Worlds" being characterized by maximal variation under a minimal set of basic principles. The goal of efficiency reflects the Darwin-Wallace principle of survival of the fittest.

The basic ingredients for the automation of CMM are

- automation of discretization of differential/integral equations,
- automation of optimization,
- automation of modeling.

Automation of the discretization of differential/integral equations is realized through *adaptive finite element methods*, with *feed-back* from computed solutions to *meshing* and (iterative) *solution of discretized equations*.

Automation of optimization, aiming at minimizing cost-functionals, is realized through feed-back from computed solutions to control variables.

Automation of modeling is realized through feed-back from computed solutions to model variables.

In all cases, adaptive computational feed-back plays a key role and thus closely couples to processes of *learning*, concerning mesh, control, and model. Furthermore, error-control, based on evaluation of residuals coupled with sensitivity information obtained by solving dual linearized equations, plays a key role.

The realization of **FEniCS** builds on cooperation between Johnson and Scott, who met as Dickson Instructors at the University of Chicago 1974–75, during the period when the revolutionary development in computational mathematics based on finite elements was initiated with the group around J. Douglas and T. Dupont at the University of Chicago playing a key role. Through the years, Johnson and Scott have independently and in cooperation developed ambitious programs in CMM now ready for fusion.

The necessary prerequisites for this venture are now available: (i) mathematical methodology, (ii) modern tools of computer science, (iii) modern computers, and (iv) crucial manpower and commitment.

More specifically, the research groups of Johnson and Scott have developed the following important elements of CMM: adaptive finite element methods connecting to (i), software such as **DOLFIN**, **Analysa** and **FIAT** connecting to (ii), and the *Body & Soul Reform Project* on CMM in applied mathematics education connecting to dissemination of results. In particular, Johnson has together with co-workers since the mid 80's developed a general methodology for adaptive error control in finite element methods, based on *duality*. This methodology is now becoming a standard in several areas of applications, but its full potential still remains to be realized. Most importantly, these research groups are now ready to join forces in a coordinated effort to reach a new standard in CMM: **FEniCS**.

3. Automation of CMM

Automation of CMM includes the automation of discretization, optimization, and modeling. Although many of the tools necessary for this automation are today available, within the prototype implementations of **FEniCS** described below, the full automation of CMM has yet to be realized in practice.

3.1. Automation of discretization.

Automation of discretization includes the automation of evaluation of variational forms, specifying differential/integral equations, with the variational form given in standard mathematical notation. It includes also the automation of meshing, automated solution of the discretized equations, and automated generation of arbitrary finite elements.

3.1.1. *Automated evaluation of variational forms.* With automated evaluation of variational forms, a differential/integral equation can be given directly in mathematical notation, which opens new possibilities for mathematical modeling; the model can be easily changed (by just specifying the new model) without the need for a new implementation. This is also a key ingredient for automation of modeling.

3.1.2. *Automated meshing.* Using adaptive finite element methods, space-time meshes can be automatically generated and adapted, to produce solutions of prescribed quality. Automated meshing enables *efficiency* of computation (minimal work for desired accuracy) and *reliability* of computed solutions (the error is less than a given tolerance).

In its most general form, automated meshing is *multi-adaptive*, meaning that the mesh is adapted without any constraints in both space and time. This means in particular that different time steps are used in different regions of space.

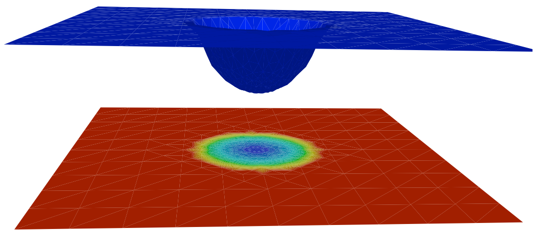


FIGURE 1. Time step size as function of space in a solution of the time-dependent heat equation with a variable heat source localized to the center of the domain. The multi-adaptive algorithm automatically uses small time steps in the active region.

3.1.3. *Automated solution of discretized equations.* Traditional methods for solving nonlinear discretized equations are based on Newton methods, with the linearized equations being solved using direct methods based on Gaussian elimination, or iterative methods such as the conjugate gradient method (CG) or GMRES. These methods are static with limited feed-back and are thus, from a computer science point of view, of limited interest.

Modern adaptive methods with more feed-back represent dynamical computational processes and pose real challenges to computer science. An example is the adaptive multigrid method, where the number of iterations on each level are adaptively determined during the iteration process. Another example is the explicit solver for stiff systems of differential equations in **DOLFIN**, where the iteration process is adaptively stabilized using a combination of large and small time steps.

3.1.4. *Automated generation of arbitrary elements.* By automating the generation of finite elements, arbitrary elements can be specified in mathematical notation. In this way, an implementation does not have to be designed for a specific element or a specific collection of elements. New elements can thus be added to the system by just giving their definition. Another advantage is that by automating the generation of elements, it becomes easier to guarantee the *correctness* of the implementation, since the implementation is common for all elements. In addition, the finite elements may be chosen adaptively by feed-back from computation.

3.2. **Automation of optimization.** With automated discretization realized in the form of computational solution of *forward problems*, a large variety of *optimization problems* or *inverse problems* may be attacked. In optimization one seeks to minimize a cost functional depending on the solution of a forward problem by varying a control variable. Typically, the minimization is achieved using a gradient method involving the solution of the forward problem and evaluation of the gradient by solving an associated dual problem. In automation of optimization, we thus rely on automation of discretization of both the forward and dual problems. We further couple optimization and discretization to achieve quantitative error control of specified output at minimal computational cost.

In many cases we formulate the optimization problem as a problem of finding a stationary point of an associated Lagrangian which takes the form of a system of differential equations, involving the forward and the dual problem, as well as an equation expressing stationarity with respect to variation of control variables. The optimization problem is thus solved by computational solution of a system of differential equations.

3.3. **Automation of modeling.** Further, with tools of automated discretization, a new world of automation of modeling opens, which may be viewed as the overall goal of CMM. In automated modeling, subgrid models are constructed through computation on resolvable scales typically through various forms of *extrapolation*. In LES of turbulence, the subgrid model may take the form of a *turbulent viscosity* which is determined locally in space and time by scale-extrapolation from computations on resolvable scales using principles of *scale similarity*, or by local in space-time resolution of all scales and extrapolation in space-time.

4. Master plan

The goal of **FEniCS** is to develop a tool of CMM with strong impact in both academics and industry. In the narrow sense, **FEniCS** may be viewed as a project for developing open-source software for automation of CMM, including a general implementation of adaptive finite element methods, but **FEniCS** has a wider potential connecting to educational reform in science and engineering.

4.1. **Software requirements.** According to the vision of **FEniCS**, the overall software requirements are *generality*, *efficiency*, and *simplicity*. These goals are generally thought of as being in contrast to one another; general codes are often inefficient and complex, and efficient codes are rarely general. However, this does not have to be the case. Simplicity can be achieved *through* generality, and, by using modern software techniques, the generality does not have to lead to decreased performance.

We also pose specific requirements concerning *correctness* and *open-source* licensing.

4.1.1. *Generality.* Generality includes the ability to handle

- general differential/integral equations,
- general meshes,
- general finite elements,
- general input and output formats following open standards.

4.1.2. *Efficiency.* Overall efficiency is obtained by

- efficiency of discretization, optimization, and modeling through adaptivity,
- efficiency of implementation, including parallelization.

4.1.3. *Simplicity.* Simplicity is obtained through a clear and modular conceptual structure, which should be reflected at all levels in the implementation of the software. This simplifies development and allows more people to be involved and contribute.

Similarly, the API to **FEniCS** shall be simple and intuitive. Specifically, a simple and flexible scripting language shall provide a Problem Solving Environment (PSE). Possibly, a Graphical User Interface (GUI) may also be provided.

4.1.4. *Correctness.* Overall correctness is obtained by

- correctness of discretization, optimization and modeling through adaptive error control,
- correctness of implementation through automatic generation of code.

4.1.5. *Open-source development.* **FEniCS** is an open-source project. Specifically, this means that **FEniCS** will be licensed under the GNU General Public License (GPL). The benefits of this are, among others,

- stability in code development,
- improvements in generality, efficiency, simplicity, and correctness,
- increased dissemination.

For further discussion, see [5].

4.2. **Reform of education.** CMM is offering new tools in science and engineering, requiring reformation of the education from basic to graduate level. The Body and Soul-project [6], involving books [7] and software, represents the first coordinated effort to meet these demands. The goal is again to set a new standard in science/engineering education and **FEniCS** has an important role to play to meet this goal. The potential impact of **FEniCS** as the computational tool in a reformed education is very strong.

5. Initial values

In this section, we specify the initial values of **FEniCS**, the current status which is the starting point for the **FEniCS** project. In particular, we give an overview of what has already been implemented in our software projects **DOLFIN**, **Analysa**, and **FIAT**, which are existing prototype implementations of the full program, and briefly discuss how these sub projects can be integrated within **FEniCS**.

5.1. **Software.**

5.1.1. **DOLFIN**. **DOLFIN** (Dynamic Object-oriented Library for FINite element computation) is a platform for adaptive finite element computation developed by Hoffman and Logg at the Department of Computational Mathematics at Chalmers. The goal of **DOLFIN** is much the same as that of **FEniCS** and it currently implements a subset of the features envisioned for **FEniCS**.

Features of **DOLFIN** include automatic evaluation of variational forms, automatic assembly from a given variational formulation, adaptive mesh refinement in two and three space dimensions, algebraic solvers such as GMRES, CG and LU, visualization using OpenDX, a system for integration of new modules/solvers for specific problems, and a multi-adaptive ODE-solver with automatic validation of solutions using solutions of automatically generated dual problems. Thus, **DOLFIN** can be thought of as a working prototype implementation of **FEniCS**.

5.1.2. **Analysa**. **Analysa** is a problem-solving environment (PSE) for partial differential equations (PDE) in two or three dimensions using the language of variational formulations. It uses `Scheme` as a scripting language, with numerous libraries incorporated automatically.

Analysa allows arbitrary variational forms to be defined, and it provides a notation for the action of these forms. The forms can have an arbitrary number of variables, and the resulting forms are evaluated efficiently both in terms of computation and storage. Arbitrary order piecewise polynomial spaces can be specified. **Analysa** uses an *embedded language* written in the functional language `Scheme` together with compiler technology to combine the expressiveness of functional languages with the efficiency of hand-coded Fortran or C. `Scheme` is simply used as a scripting language in this context. **Analysa** snarfs many libraries for various purposes, including but not limited to graphics, windowing, linear algebra, domain geometry, numerical quadrature, and finite elements.

5.1.3. **FIAT**. **FIAT** (FInite element Automatic Tabulator) is a recently-developed Python package that tackles the question of computing arbitrary finite elements. Mathematically, a finite element can be defined as a domain plus a finite-dimensional function space and an associated set of linear functionals (nodes). Many finite element implementations are constrained by our ability to obtain explicit formulae for the basis functions. **FIAT** removes this constraint by using linear algebra and higher order programming techniques to construct the nodal bases for arbitrary elements, as linear combinations of orthogonal polynomials. **FIAT** may not currently be efficient enough to be used in a run-time system, but is effective at pre-computing basis function values to be fed into a solver.

5.1.4. **FEniCS**. Although **DOLFIN**, **Analysa**, and **FIAT** are three separate implementations approximating the functionality of **FEniCS**, they are in a sense orthogonal, which means that they combine easily to a new and better approximation of **FEniCS**. The first approximation of **FEniCS** will be to use **DOLFIN** as the computational kernel, and write an interface between **DOLFIN** and **FIAT** so that the elements generated by **FIAT** are used within **DOLFIN**. On top of this a simple scripting language in the style of **Analysa** is

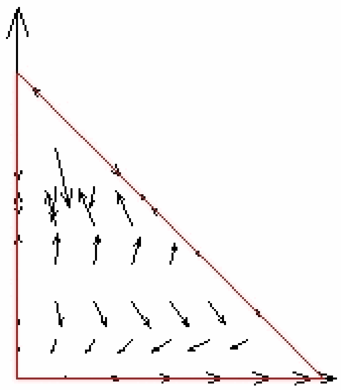


FIGURE 2. A fifth order Raviart–Thomas basis function generated by **FIAT**.

placed, thus obtaining the first version of **FEniCS**. From there on the development can continue with better approximations converging to **FEniCS**.

5.2. **Man-power.** The **FEniCS** core team consists of the following people, committed to implementing a new standard in CMM software:

- Prof. Todd Dupont
  Dept. of Computer Science
  University of Chicago, USA
- Dr. Johan Hoffman
  Courant Institute of Mathematical Sciences, New York, USA
- Prof. Claes Johnson
  Dept. of Computational Mathematics
  Chalmers University of Technology, Göteborg, Sweden
- Ass. Prof. Robert C. Kirby
  Dept. of Computer Science
  University of Chicago, Ill USA
- Ass. Prof. Mats Larson
  Dept. of Computational Mathematics
  Chalmers University of Technology, Göteborg, Sweden
- Mr. Anders Logg
  Dept. of Computational Mathematics
  Chalmers University of Technology, Göteborg, Sweden

• Prof. Ridgway Scott
Dept. of Computer Science
University of Chicago, USA

The project will be based at the Toyota Technological Institute (TTI) at Chicago and the Department of Computer Science at the University of Chicago, and will be run in cooperation with Chalmers University of Technology in Göteborg.

Further, the project may be expected to couple to activities within Computer Science at TTI/University of Chicago, such as learning and compilers.

6. Challenges

CMM has brought revolutionary new tools to science and industry, but there are many challenges requiring further development, of which we list a few below. Basic open problems concern multi-scale problems with computationally unresolvable scales in space and time.

A specific goal of **FEniCS** is to open new possibilities for attacking such problems, all requiring automation of CMM. We believe that **FEniCS** will enable substantial progress to be made for a large variety of problems in science and industry in general, and for the problems below in particular.

6.1. **Turbulence modeling.**

Turbulent flow at high Reynold's numbers such as 10^6 typically requires on the order of 10^{18} arithmetic operations in *Direct Numerical Simulation* (DNS), using standard computational strategies, which is way beyond foreseeable computational power.

In *Large Eddy Simulation* (LES), the larger eddies are resolved in computations with say 10^6 mesh points and the effect of unresolved smaller eddies is modeled in *subgrid models* in terms of the larger eddies. The challenge is to design such subgrid models allowing accurate computation of mean values on resolvable scales.

Today, new possibilities to meet this challenge are opened through computational modeling, where subgrid models are constructed adaptively from computations on resolvable scales. One may view computational turbulence modeling as a *learning process*, where the subgrid model learns how to model turbulence.

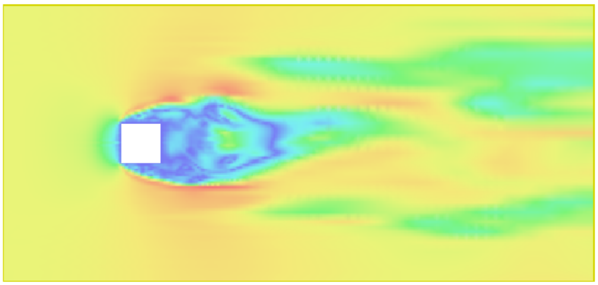


Figure 3. Cross-section of 3D turbulent flow around a surface mounted cube. (LES using **DOLFIN**.)

6.2. **Protein folding.** Another basic problem requiring subgrid modeling is that of *molecular dynamics*, where the shortest time scale of 10^{-15} seconds is unresolvable in simulations of e.g. protein folding which require simulation time intervals up to seconds. Successful such simulations may give important insight into the function of proteins in life processes.

6.3. **Quantum mechanics.** Another challenge concerns computational solution of *Schrödingers equations* in *quantum mechanics*, involving high-dimensional differential equations, again requiring computational modeling for solvability. Computational quantum mechanics is the basis for computational chemistry and biology.

6.4. **Design.** CMM opens entirely new possibilities in production design by replacing experimental testing by computational simulation. For example, crash simulation for car design is today a standard tool, but many expensive crash tests are still performed. Another example is the design of antennas for tele-communication.

6.5. **Inverse problems.** There is a variety of *inverse problems* in e.g. geophysics and medical imaging, where the objective is to determine properties inside a body from measurements on parts of the surface of the body, which in mathematical terms corresponds to determining coefficients in differential equations from boundary data.

References

[1] J. Hoffman and A. Logg, **DOLFIN**,
`http://www.phi.chalmers.se/dolfin/`

[2] B. Bagheri and R. Scott, **Analysa**,
`http://people.cs.uchicago.edu/~ridg/al/aa.html`

[3] R. Kirby, *FIAT: A new paradigm for computing finite element basis functions*, submitted to Transactions on Mathematical Software.

[4] A. Logg, *Tanganyika, a multi-adaptive ODE-solver*,
`http://www.phi.chalmers.se/tanganyika`

[5] *The GNU project*, `http://www.gnu.org/`

[6] K. Eriksson, D. Estep, C. Johnson, *The Body & Soul reform project in applied mathematics education*,
`http://www.phi.chalmers.se/bodysoul/`

[7] K. Eriksson, D. Estep, C. Johnson, T. Barth, J. Hoffman, A. Logg, *Applied Mathematics: Body & Soul, Vol 1–5*, Springer-Verlag, 2003–04.

[8] S. Brenner and and L.R. Scott *The Mathematical Theory of Finite Element Methods, 2nd Ed.* Springer-Verlag, 2002.

[9] L.R. Scott and B. Bagheri Software environments for the parallel solution of partial differential equations. In *Computing Methods in Applied Sciences and Engineering IX, R. Glowinski and A. Lichnewsky, eds.* (1990), Philadelphia: SIAM, pp. 378–392.

APPENDIX

— Appendix A: **DOLFIN**
— Appendix B: **Analysa**

Appendix A. DOLFIN

A.1. **Background.** The **DOLFIN** project [1] was initiated in 2002 with the intention to provide a common platform for development and research at the Department of Computational Mathematics at Chalmers and the interdisciplinary Chalmers Finite Element Center Φ, building on earlier experience. **DOLFIN** is the primary development platform for all graduate students at the department and is used in various courses at Chalmers, ranging from first-year undergraduate to graduate level.

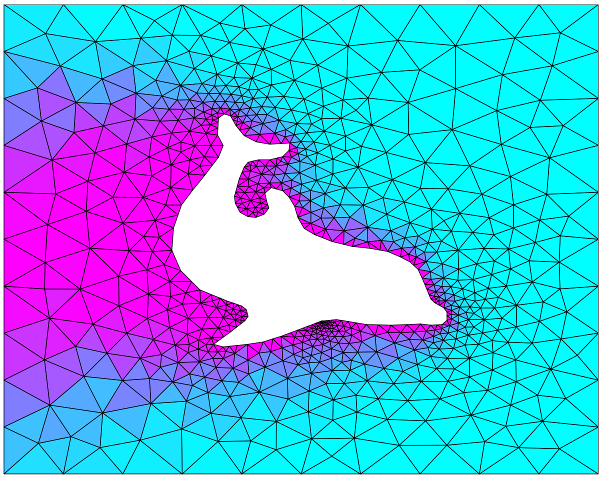


Figure 4. Convection–diffusion around a hot dolphin.

DOLFIN is implemented as a `C++` library and can be used either as a stand-alone solver, or as a tool for development and implementation of new methods. The GNU autotools (`automake` and `autoconf`) are used to automate building on different systems.

To simplify usage and emphasize structure, **DOLFIN** is organized into three levels of abstraction, which we denote by *kernel level*, *module level*, and *user level*. Core features, such as automatic evaluation of forms and adaptive mesh refinement, are implemented as basic tools at kernel level. At module level, new solvers/modules that use the basic tools as building blocks can be easily integrated into the system. At user level, a problem can be solved, either using one of the built-in solvers/modules or by using the basic tools. To make sure that each part of the system can be replaced, the code is completely modularized (see Figure 5).

We summarize the key features of **DOLFIN** as follows:

- automatic evaluation of variational forms,
- automatic assembly from variational formulation,
- adaptive mesh refinement for triangular or tetrahedral meshes,
- linear algebra including algebraic solvers,
- graphical visualization using OpenDX,
- a system for integration of new modules/solvers for specific PDEs,

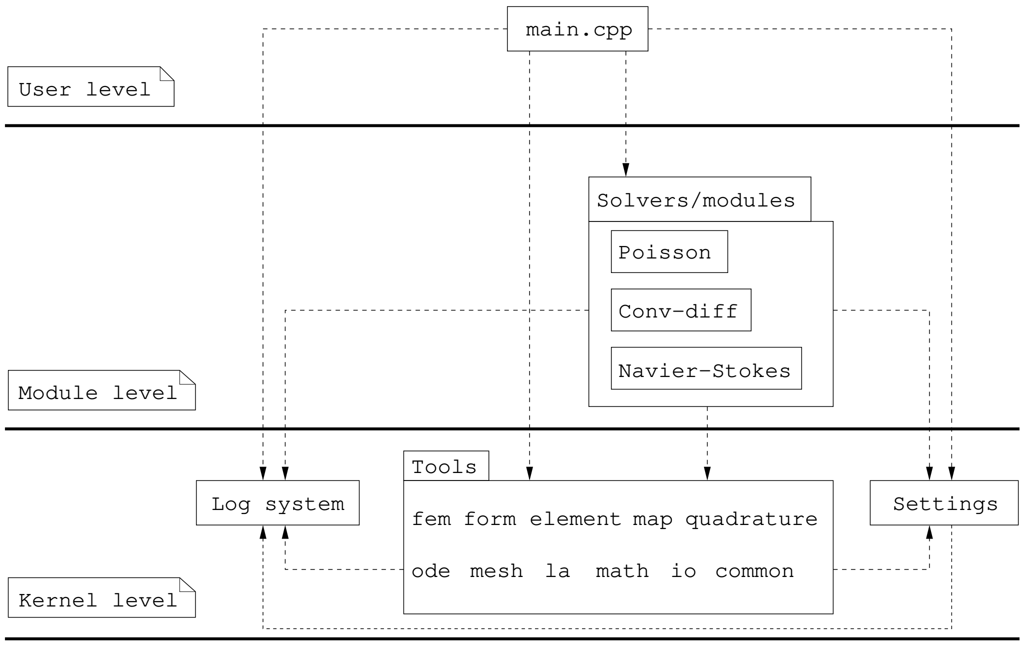


FIGURE 5. A diagram showing the three levels of abstraction and the modularization within **DOLFIN**.

- a fully automated multi-adaptive ODE-solver.

A.2. **Automatic evaluation of variational forms.** Automatic evaluation of variational forms is implemented by operator overloading in C++, allowing simple specification of variational forms in a language that is close to the mathematical notation. Good performance is obtained by automatic precomputation and tabulation of integrals, in combination with special techniques to avoid object construction.

As an example, consider Poisson's equation,

$$-\Delta u(x) = f(x), \quad x \in \Omega,$$

the variational formulation (ignoring boundary conditions) of which is

$$\sum_K \int_K (\nabla u, \nabla v)\, dK = \sum_K \int_K f v\, dK.$$

The two variational forms of the left- (`lhs`) and right-hand sides (`rhs`) are specified in **DOLFIN** as shown in Program A.1.

A.3. **Automatic assembly from variational formulation.** In **DOLFIN**, automation is provided also for the assembly of matrices and vectors from a given variational formulation, using the automatic evaluation of variational forms. This automates a large part of the implementation of a solver. In the case of Poisson's equation, the algorithm becomes particularly simple: assemble a matrix (*the stiffness matrix*) and a vector (*the load vector*),

```
class Poisson : public PDE {
  ...
  real lhs(const ShapeFunction& u, const ShapeFunction& v)
  {
    return (grad(u),grad(v)) * dK;
  }

  real rhs(const ShapeFunction& v)
  {
    return f * v * dK;
  }
  ...
};
```

Program A.1: Specification of the variational formulation of Poisson's equation in **DOLFIN**.

solve the linear system, and save the solution. This is illustrated in Program A.2, which contains the full implementation of the Poisson solver/module in **DOLFIN**.

```
void PoissonSolver::solve()
{
  Matrix   A;
  Vector   x, b;
  Function u(mesh, x), f(mesh, "source");
  Poisson  poisson(f);
  Galerkin fem;

  fem.assemble(poisson, mesh, A, b);
  A.solve(x, b);

  File file("poisson.m");
  file << u;
}
```

Program A.2: The implementation of the Poisson solver in **DOLFIN**.

Automation of all parts of **DOLFIN** makes the implementation simple and clear at all levels, not only at the top level. In fact, the algorithm for automated assembly is just a couple of lines. Program A.3 shows the full algorithm of the assembly of a matrix from a given form.

A.4. **Adaptive mesh refinement.** The concept of a *mesh* is central in the implementation of adaptive finite element methods for partial differential equations. This and other important concepts are implemented as `C++` classes in **DOLFIN**:

```
for (CellIterator cell(mesh); !cell.end(); ++cell) {

  mapping->update(cell);
  element->update(mapping);

  pde.updateLHS(element, cell, mapping, quadrature);

  for (FiniteElement::TestFunctionIterator v(element); !v.end(); ++v)
    for (FiniteElement::TrialFunctionIterator u(element); !u.end(); ++u)
      A(v.dof(cell), u.dof(cell)) += pde.lhs(u,v);

}
```

Program A.3: Automatic assembly of a matrix `A` from a given form.

- `Mesh`
- `Node`, `Cell`, `Edge`, `Face`
- `Boundary`
- `MeshHierarchy`

In addition, iterators such as `NodeIterator`, `CellIterator`, `EdgeIterator`, `FaceIterator`, and `MeshIterator` are implemented to allow easy access to the mesh information that is needed in the implementation of adaptive finite element methods, such as the node neighbors of a node or the edges within a face. In Program A.4 we give an example of a code that displays all the node neighbors of the nodes of all the cells within a given mesh.

```
for (CellIterator c(m); !c.end(); ++c)
  for (NodeIterator n1(c); !n1.end(); ++n1)
    for (NodeIterator n2(n1); !n2.end(); ++n2)
      cout << *n2 << endl;
```

Program A.4: Iteration over all node neighbors `n2` of the nodes `n1` within all cells `c` of the mesh `m`.

Adaptive mesh refinement for triangular meshes (in 2D) and tetrahedral meshes (in 3D) is implemented in **DOLFIN**, see Figure 6. Refining a mesh is simple, just mark the cells (according to some criterion for refinement) and tell the mesh to refine itself, see Program A.5. A hierarchy of meshes, that can be used for multigrid computation, is automatically created.

A.5. **Linear algebra.** **DOLFIN** includes an efficient implementation of the basic concepts of linear algebra: matrices and vectors. Both sparse and dense matrices are implemented, as well as generic matrices only defined through their action (multiplication with a given vector).

Several algebraic solvers are implemented in **DOLFIN**. These include preconditioned iterative methods such as GMRES and CG, and direct methods such as LU factorization.

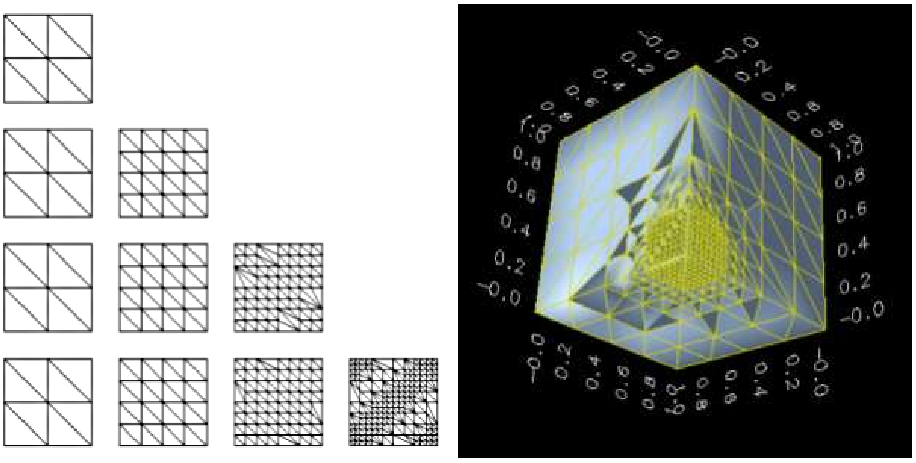


FIGURE 6. Adaptive mesh refinement of triangular and tetrahedral meshes within **DOLFIN**.

```
File file(``mesh.xml''); // Create a file
Mesh mesh;               // Create a mesh

// Read mesh from file
file >> mesh;

// Mark cells for refinement
for (CellIterator cell(mesh); !cell.end(); ++cell)
  if ( ... )
    cell->mark();

// Refine mesh
mesh.refine();
```

Program A.5: Code for adaptive mesh refinement in **DOLFIN**.

A.6. **Graphical visualization.** **DOLFIN** contains no built-in visualization and relies on interaction with external tools for visualization, such as the open-source program *OpenDX*. Post-processing (as well as pre-processing) is thus accomplished by implementation of the file formats needed for exchange of data. Using a modular approach, **DOLFIN** has been designed to allow easy extension by addition of new file formats.

A.7. **Easy integration of new modules.** A system for easy integration of new solvers/modules for specific problems has been implemented in **DOLFIN**. The hope is that people continuously will contribute by implementing their solvers as a module which will add to the functionality of **DOLFIN**.

PDE solvers in two and three dimensions for general convection–diffusion–reaction problems are implemented. A solver for the incompressible Navier–Stokes equations, which is able to handle free and moving boundaries, including LES and various subgrid models, is currently being implemented.

A.8. **Multi-adaptive ODE-solver.** A general multi-adaptive ODE-solver, including automatic generation and solution of dual problems, automatic error estimation, and adaptivity, is currently being implemented in **DOLFIN**, as the a result of a merge with the existing project *Tanganyika* [4].

A.9. **Other features.** Other important features implemented in **DOLFIN** include:

- multigrid (in preparation),
- log system.
- parameter management,

A multigrid solver is currently being implemented in cooperation with graduate students at the Department of Computational Mathematics at Chalmers.

A uniform log system for event notification, including diagnostic messages, warnings and progress status, is used throughout the code. This gives flexibility as to the way in which information is formatted and displayed. The current log system includes both a simple terminal-based and curses-based (graphical) format.

Parameter management, such as specification of tolerances and coefficients, is handled by a central database of parameters, which can be dynamically added, changed and retrieved.

Appendix B. **Analysa**

B.1. **Background.** **Analysa** [2] was based on a finite element library called `fec` [9]. **Analysa** uses a particular embodiment of `Scheme` called `alscheme`. This augments `Scheme` by adding an *object system* called `tinyClos`, and this allows one to program in a way similar to what is often done in `C++`. In addition, `alscheme` implements constraints, and it incorporates the library `slib` which provides many convenient features. Consult the `alscheme` manual for details.

B.2. **Form notation.** Variational forms are defined in a list following the key word `integral-forms`. The first example in Program B.1 is the "mass" inner product form, followed by the standard "stiffness" form `k` for Laplace's equation, etc.

```
(parameters
  (alpha 1.0)
  (nu 1.0)
  (rho 10000.0)
)
(integral-forms
  ((m u v) (* u v))
  ((k u v) (dot (gradient u) (gradient v)))
  ((d u v) (* (divergence u) (divergence v)))
  ((p u v) (+ (* alpha (m u v))
              (* nu (k u v))
              (* rho (d u v))))
)
```

Program B.1: Examples of four variational forms and some associated parameters in **Analysa**.

The variational form itself is the integral of the specified formula over the domain of `u`, which is assumed implicitly to be the same as the domain of `v`. The key-word operators `gradient`, `divergence`, `dot` and so forth correspond to multi-variate calculus concepts. The parameters `alpha`, `nu`, etc., would have been defined in a prior statement as shown in Program B.1.

B.3. **Piecewise-polynomial interpolants.** In **Analysa**, finite-dimensional spaces of functions are used to approximate functions defined by differential equations or other implicit definitions. In many cases, the function approximations can be thought of as "interpolating" the function being approximated.

Finite elements have both a local version (the *element*) and a global incarnation which involves piecing together copies of the local element on a mesh. The local part is defined in **Analysa** by a keyword. **Analysa** does not provide a mechanism for specifying new elements. However, it does allow for a parameterized element type. At the moment, the only element type supported is the Lagrange element in two and three dimensions. The degree of the polynomials can be arbitrary, though. An example of the notation is

```
(elements
   (elagrans (lagrange-simplex 7))
)
```

which constructs an element of degree seven. The dimension of the underlying space is implicitly determined from the mesh when the element is used.

B.4. **Finite element space.** The element and mesh are put together via the built-in operator `fe` in **Analysa** which creates the function space. Again, there is a *define* operator `spaces` followed by a list of pairs `(a b)` where `a` is the name of the space and `b` describes how it is constructed. The specific notation is

```
(spaces
  (globulspas (fe elagrans (all RT-mesh) r^2:))
  (boundryspas (fe elagrans (boundary RT-mesh) r^2:))
  (interior-spase (fe elagrans ((- all boundary) RT-mesh) r^2:))
)
```

This links the "reference element" `elagrans` with the mesh `RT-mesh` being used to create the space of *piecewise*-polynomial Lagrange interpolants with respect to the mesh `RT-mesh`. We see that this will be a space of functions defined on two-dimensional space (because the mesh comes from such a domain) with vector values (indicated by the notation `r^2:`). Similarly,

```
(global-spice (fe elagrans (all RT-mesh) r:))
```

defines a space of functions defined on the same domain in two-dimensional space with scalar values.

The basic space constructor `fe` is a function of three variables: the element, the mesh, and the value space. Its result is the space of interpolants based on that local element, defined on that mesh, with corresponding values of the type specified.

These spaces of functions are defined on the different meshes. There are built-in operators on meshes which allow one to work with functions defined on sub-meshes. For example, `(all RT-mesh)` means that values at *all* mesh points should be used, whereas `(boundary RT-mesh)` indicates that only *boundary* points should be used. The calculus

```
(- all boundary)
```

specifies the interior mesh points by taking the (set-theoretic) difference of `all` and `boundary`.

The approach of linking a local definition of the interpolation process with a global subdivision (collection of pieces which can be mapped to the local domain) comes from the standard implementation technique for the *finite element method* [8].

Suppose `rhsf` is a function given by a product of "sine" functions. Then the interpolant of a this function is defined by the code in Program B.2.

```
(functions
  (rhsf (interpolant globulspas (lambda (v)
                       (let ((x (_ v 0)) (y (_ v 1)))
                          (* (sin x) (sin y))
                        ))))
)
```

Program B.2: Code to define a scalar-valued interpolant in **Analysa**.

B.5. **Projections.** It is necessary to work with spaces defined on different domains in a boundary value problem. The basic equations for the Dirichlet problem involve determination of values of an interpolant in the interior, but if data is non-homogeneous, a specification of it on the boundary will be needed.

Suppose that we define a global space `a` using the `all` constructor for space, a boundary space `b` using the `boundary` constructor, and `c` for the interior space using the `(- all boundary)` constructor. Similarly let us suppose we have interpolants `u` in `a`, `v` in `b`, and `w` in `c`. Then we clearly must do something to combine `u`, `v` and `w` since they are not in the same spaces. But they are in *related* spaces. Thus, it makes sense say to add a projection of one to another, e.g.,

```
(+ w (projection u c))
```

which defines a new function in `c`. This makes good sense since the space `c` is contained in the space `a`. But it is also possible to *extend* something from `b` to `a` just by setting the values in the interior to zero. Thus

```
(+ u (projection v a))
```

is defined in **Analysa** as well. The extension

```
(projection w a)
```

is also done by setting values on the boundary to zero. Note that

```
(projection  (projection v a) c)
```

would always be zero, since we extend by zero and then restrict to the interior.

B.6. **Form actions.** In many cases, you do not want to evaluate the matrix (or tensor) associated with a form at all. The *action* of a form may be all that is required. Let us begin by defining this for a bilinear form $a(\cdot, \cdot)$.

Suppose that we have a space S with basis $\{\varphi_i \,:\, i = 1, \ldots, N\}$, and define the corresponding matrix K whose entries are $a(\varphi_i, \varphi_j)$. Then we can express matrix multiplication of the form $V = KU$ in terms of what it does to elements of the space S as follows. Let u correspond to U ($u := \sum_i U_i \varphi_i$) and let v correspond to V ($v := \sum_i V_i \varphi_i$). The components of V are of course defined by

$$V_i = \sum_j a(\varphi_i, \varphi_j) U_j = a(\varphi_i, u) \tag{B.1}$$

for all $j = 1, \ldots, N$. Thus, we suggest the notation

$$v = a(S, u) \tag{B.2}$$

to express the same meaning. Correspondingly, **Analysa** implements

$$K = a(S, S) \tag{B.3}$$

to define the matrix K. Thus, we have a hierarchy of tensors that can be defined from a form: $a(u, v)$ is a number, $a(u, S)$ is a vector, and $a(S, S)$ is a matrix. This is just notation, but it is quite suggestive and unambiguous. Note that $a(u, S)$ may be different from $a(S, u)$. Also, forms may be defined on Cartesian products of different spaces, not just copies of the same space. Thus, you might have a matrix given by $a(S^1, S^2)$ for two different spaces S^1 and S^2.

To illustrate the use of the form action, let us suppose that we have an additional form $m(u, v)$ and that we wish to solve an equation of the form

$$KU = MF \tag{B.4}$$

where $f = \sum_i F_i \varphi_i$. We can do this in two ways. We can define an interpolant $\phi = m(f, S)$ and then provide this to the appropriate solution routine. Or we could define $M = m(S, S)$ and compute the matrix-vector product MF explicitly. In the former case, it is not necessary to compute the matrix M at all.

The notation for general multi-linear forms is the same. Thus $c(u, u, S)$ is a vector. Its definition and evaluation do not require that a high-order tensor be evaluated and stored. In solving non-linear problems, there is never the need to have the full tensor evaluated. Only the actions on specific interpolants are required. However, in certain cases a matrix such as $c(S, u, S)$ might be of interest.

B.7. **Solvers.** **Analysa** provides several built-in solvers for linear problems, but it also allows general solvers to be described using `Scheme`. Suppose that we have defined a form `pf`, a space `int-spas` and a member `rhs` of that space. Then

```
(define p-matrx (pf int-spas int-spas))
(lu-factor! p-matrx)
(set-values! u (lu-solve p-matrx rhs))
```

forms the matrix `p-matrix` corresponding to the form `pf`, factors it, and solves using these factors (via sparse Gaussian elimination using minimum degree ordering). The result has been updated in `u`.

A solver for the Stokes equations is presented in Program B.3. It involves an iteration in which repeated `lu-solve`s are done on a fixed matrix `p-matrix` (which gets factored only once). However, the equation being solved is not just the one associated with `p-matrix`. The solution involves satisfying the divergence constraint. This gets imposed by the form `divform`. Note that at each iteration, the action of `divform` is computed on `u` to yield `du`. For details, see [8]. The "format" statements come from `slib` and provide C-like printing functionality.

```
(solvers
  (penalty
   (set-values! u (projection bc global-space))
   (set-values! w (projection bc global-space))
   (set-values! dw (projection bc global-space))
   (define k 0)
   (define unorm (norm2 u))
   (define dunorm unorm)
   (define p-matrix (p interior-space interior-space))
   (set-values! rhs (- (projection (m global-space f) interior-space)
             (projection (p global-space (projection bc global-space))
                                   interior-space)))
   (lu-factor! p-matrix)
   (format #t "# of elements in factored p-matrix: ~a~%"
     (+ (math.no-of-original-elements p-matrix)
        (math.no-of-fillin-elements p-matrix)))
   (while (and (< k max-iters)
               (> dunorm (* tolerance unorm)))
     (set-values! u (+ (projection
          (lu-solve p-matrix (+ rhs (projection dw interior-space)))
                 global-space)
          (projection bc global-space)))
     (set! unorm (norm2 (projection u interior-space)))
     (set-values! du (divform u global-space))
     (set! dunorm (norm2 (projection du interior-space)))
     (set-values! w (- w (* rho u)))
     (set-values! dw (- dw (* rho du)))
     (set! k (+ k 1))
     (format #t " penalty solver: iteration ~a, divergence ~a~%"
                       k (/ dunorm unorm))
   )
  )
)
```

Program B.3: A solver for the Stokes equations in **Analysa**.

B.8. **Graphical visualization.** **Analysa** provides a window-based display for visualizing the results. The things you can see are controlled by a `displays` list of the form

```
(displays
  (mesh gm)
  (all g u)
)
```

The command

```
(refresh all)
```

causes a display to be updated on the screen.

Analysa chooses particular display types automatically depending on the value space for a given interpolant. Thus a vector-valued interpolant is displayed using a vector plot. A scalar-valued function is done differently. Three-dimensional graphics is of course quite complex. **Analysa** implements some slicing capability for enhanced visualization in three dimensions.

B.9. **Numerical quadrature and variable coefficients.** **Analysa** utilizes a type of Gaussian quadrature that is exact for polynomials on triangles and tetrahedra. When there are coefficients to be included in a form, there is the need to integrate non-polynomial functions. There are different philosophies on how to handle this. **Analysa** takes the point of view that all approximations should be done in explicit spaces. So a variable coefficient would first be interpolated into a space (but perhaps a higher-order space). Then the variable coefficient form is represented simply as a multi-linear form with the coefficient being replaced by its interpolant.

Chalmers Finite Element Center Preprints

2001–01 *A simple nonconforming bilinear element for the elasticity problem*
Peter Hansbo and Mats G. Larson

2001–02 *The $\mathcal{LL}^*$ finite element method and multigrid for the magnetostatic problem*
Rickard Bergström, Mats G. Larson, and Klas Samuelsson

2001–03 *The Fokker-Planck operator as an asymptotic limit in anisotropic media*
Mohammad Asadzadeh

2001–04 *A posteriori error estimation of functionals in elliptic problems: experiments*
Mats G. Larson and A. Jonas Niklasson

2001–05 *A note on energy conservation for Hamiltonian systems using continuous time finite elements*
Peter Hansbo

2001–06 *Stationary level set method for modelling sharp interfaces in groundwater flow*
Nahidh Sharif and Nils-Erik Wiberg

2001–07 *Integration methods for the calculation of the magnetostatic field due to coils*
Marzia Fontana

2001–08 *Adaptive finite element computation of 3D magnetostatic problems in potential formulation*
Marzia Fontana

2001–09 *Multi-adaptive galerkin methods for ODEs I: theory & algorithms*
Anders Logg

2001–10 *Multi-adaptive galerkin methods for ODEs II: applications*
Anders Logg

2001–11 *Energy norm a posteriori error estimation for discontinuous Galerkin methods*
Roland Becker, Peter Hansbo, and Mats G. Larson

2001–12 *Analysis of a family of discontinuous Galerkin methods for elliptic problems: the one dimensional case*
Mats G. Larson and A. Jonas Niklasson

2001–13 *Analysis of a nonsymmetric discontinuous Galerkin method for elliptic problems: stability and energy error estimates*
Mats G. Larson and A. Jonas Niklasson

2001–14 *A hybrid method for the wave equation*
Larisa Beilina, Klas Samuelsson and Krister Åhlander

2001–15 *A finite element method for domain decomposition with non-matching grids*
Roland Becker, Peter Hansbo and Rolf Stenberg

2001–16 *Application of stable FEM-FDTD hybrid to scattering problems*
Thomas Rylander and Anders Bondeson

2001–17 *Eddy current computations using adaptive grids and edge elements*
Y. Q. Liu, A. Bondeson, R. Bergström, C. Johnson, M. G. Larson, and K. Samuelsson

2001–18 *Adaptive finite element methods for incompressible fluid flow*
Johan Hoffman and Claes Johnson

2001–19 *Dynamic subgrid modeling for time dependent convection–diffusion–reaction equations with fractal solutions*
Johan Hoffman

2001–20 *Topics in adaptive computational methods for differential equations*
Claes Johnson, Johan Hoffman and Anders Logg

2001–21 *An unfitted finite element method for elliptic interface problems*
Anita Hansbo and Peter Hansbo

2001–22 *A P^2–continuous, P^1–discontinuous finite element method for the Mindlin-Reissner plate model*
Peter Hansbo and Mats G. Larson

2002–01 *Approximation of time derivatives for parabolic equations in Banach space: constant time steps*
Yubin Yan

2002–02 *Approximation of time derivatives for parabolic equations in Banach space: variable time steps*
Yubin Yan

2002–03 *Stability of explicit-implicit hybrid time-stepping schemes for Maxwell's equations*
Thomas Rylander and Anders Bondeson

2002–04 *A computational study of transition to turbulence in shear flow*
Johan Hoffman and Claes Johnson

2002–05 *Adaptive hybrid FEM/FDM methods for inverse scattering problems*
Larisa Beilina

2002–06 *DOLFIN - Dynamic Object oriented Library for FINite element computation*
Johan Hoffman and Anders Logg

2002–07 *Explicit time-stepping for stiff ODEs*
Kenneth Eriksson, Claes Johnson and Anders Logg

2002–08 *Adaptive finite element methods for turbulent flow*
Johan Hoffman

2002–09 *Adaptive multiscale computational modeling of complex incompressible fluid flow*
Johan Hoffman and Claes Johnson

2002–10 *Least-squares finite element methods with applications in electromagnetics*
Rickard Bergström

2002–11 *Discontinuous/continuous least-squares finite element methods for elliptic problems*
Rickard Bergström and Mats G. Larson

2002–12 *Discontinuous least-squares finite element methods for the Div-Curl problem*
Rickard Bergström and Mats G. Larson

2002–13 *Object oriented implementation of a general finite element code*
Rickard Bergström

2002–14 *On adaptive strategies and error control in fracture mechanics*
Per Heintz and Klas Samuelsson

2002–15 *A unified stabilized method for Stokes' and Darcy's equations*
Erik Burman and Peter Hansbo

2002–16 *A finite element method on composite grids based on Nitsche's method*
Anita Hansbo, Peter Hansbo and Mats G. Larson

2002–17 *Edge stabilization for Galerkin approximations of convection–diffusion problems*
Erik Burman and Peter Hansbo

2002–18 *Adaptive strategies and error control for computing material forces in fracture mechanics*
Per Heintz, Fredrik Larsson, Peter Hansbo and Kenneth Runesson

2002–19 *A variable diffusion method for mesh smoothing*
J. Hermansson and P. Hansbo

2003–01 *A hybrid method for elastic waves*
L.Beilina

2003–02 *Application of the local nonobtuse tetrahedral refinement techniques near Fichera-like corners*
L.Beilina, S.Korotov and M. Křížek

2003–03 *Nitsche's method for coupling non-matching meshes in fluid-structure vibration problems*
Peter Hansbo and Joakim Hermansson

2003–04 *Crouzeix–Raviart and Raviart–Thomas elements for acoustic fluid–structure interaction*
Joakim Hermansson

2003–05 *Smoothing properties and approximation of time derivatives in multistep backward difference methods for linear parabolic equations*
Yubin Yan

2003–06 *Postprocessing the finite element method for semilinear parabolic problems*
Yubin Yan

2003–07 *The finite element method for a linear stochastic parabolic partial differential equation driven by additive noise*
Yubin Yan

2003–08 *A finite element method for a nonlinear stochastic parabolic equation*
Yubin Yan

2003–09 *A finite element method for the simulation of strong and weak discontinuities in elasticity*
Anita Hansbo and Peter Hansbo

2003–10 *Generalized Green's functions and the effective domain of influence*
Donald Estep, Michael Holst, and Mats G. Larson

2003–11 *Adaptive finite element/difference method for inverse elastic scattering waves*
L.Beilina

2003–12 *A Lagrange multiplier method for the finite element solution of elliptic domain decomposition problems using non-matching meshes*
Peter Hansbo, Carlo Lovadina, Ilaria Perugia, and Giancarlo Sangalli

2003–13 *A reduced P^1–discontinuous Galerkin method*
R. Becker, E. Burman, P. Hansbo, and M. G. Larson

2003–14 *Nitsche's method combined with space–time finite elements for ALE fluid–structure interaction problems*
Peter Hansbo, Joakim Hermansson, and Thomas Svedberg

2003–15 *Stabilized Crouzeix–Raviart element for the Darcy-Stokes problem*
Erik Burman and Peter Hansbo

2003–16 *Edge stabilization for the generalized Stokes problem: a continuous interior penalty method*
Erik Burman and Peter Hansbo

2003–17 *A conservative flux for the continuous Galerkin method based on discontinuous enrichment*
Mats G. Larson and A. Jonas Niklasson

2003–18 *CAD–to–CAE integration through automated model simplification and adaptive modelling*
K.Y. Lee, M.A. Price, C.G. Armstrong, M.G. Larson, and K. Samuelsson

2003–19 *Multi-adaptive time integration*
Anders Logg

2003–20 *Adaptive computational methods for parabolic problems*
Kenneth Eriksson, Claes Johnson, and Anders Logg

2003–21 *The FEniCS project*
T. Dupont, J. Hoffman, C. Johnson, R. Kirby, M. Larson, A. Logg, and R. Scott

These preprints can be obtained from

`www.phi.chalmers.se/preprints`