Current status of DOLFIN

A guide for prospective developers

Johan Hoffman & Anders Logg

`dolfin@math.chalmers.se`

Department of Computational Mathematics

Outline

- Overview
- Input / output
- A very quick guide to C++
- Development with DOLFIN
- Summary of features

Overview

Introduction

- An adaptive finite element solver for PDEs (and ODEs)
- Written by people at the Department of Computational Mathematics (Hoffman/Logg)
- Written in C++
- Only a solver. No grid generation. No visualisation.
- Licensed under the GNU GPL
- `http://www.phi.chalmers.se/dolfin`

Evolution of DOLFIN

- First public version, 0.2.6, was released Feb 2002.
- The latest version, 0.3.10, was released Sep 2003.
- The main part of the code has been written by Hoffman/Logg but contains contributions from 8 other people.
- At `www.freshmeat.net`, 13 people are listed as subscribers to new versions of DOLFIN.
- The latest CVS version consists of 35.000 lines of code.

As you can see, DOLFIN is yet quite a small project, but we hope to grow! (Not necessarily in terms of *klocs*. . .)

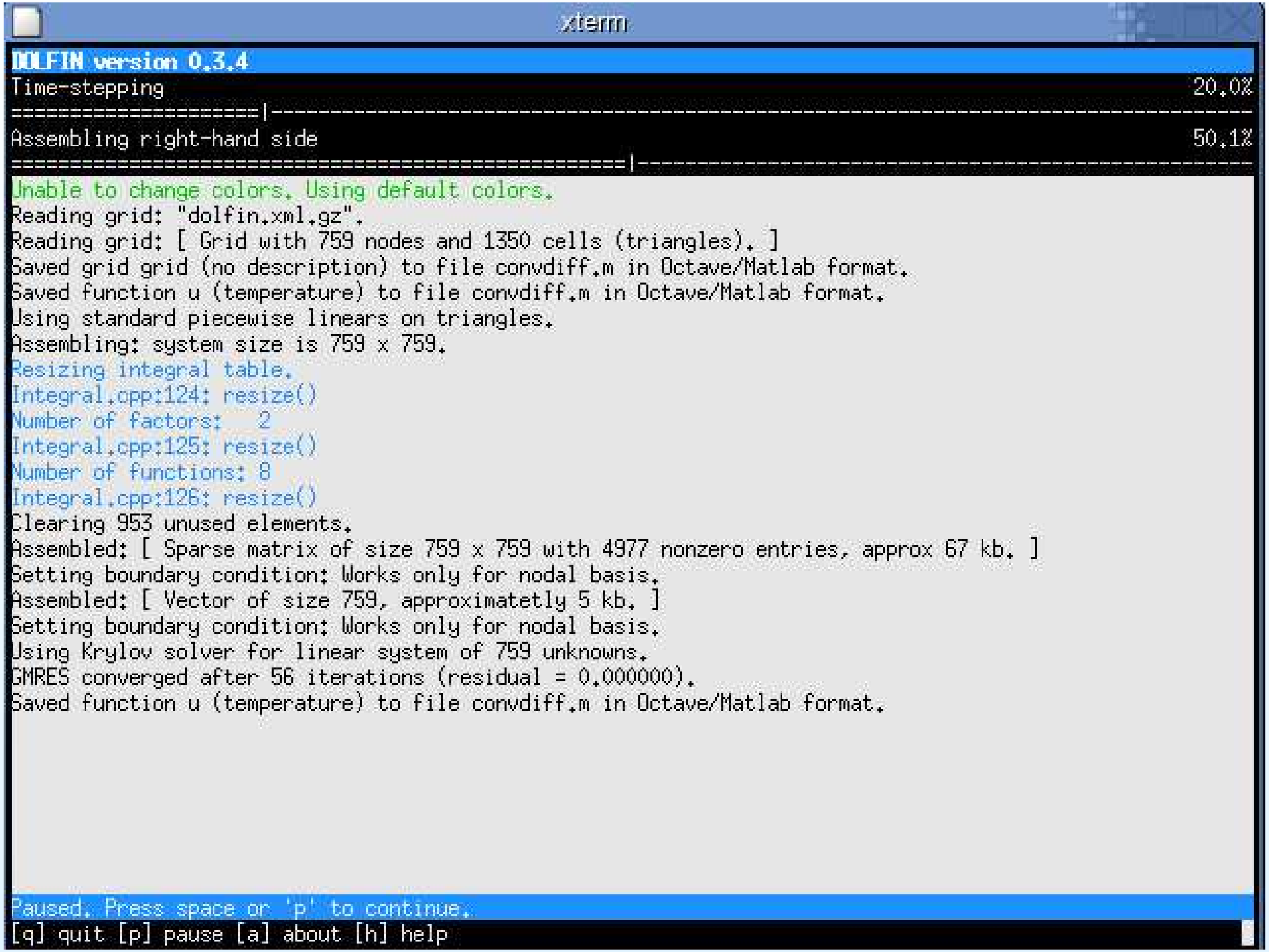
GNU and the GPL

- Makes the software free for all users
- Free to modify, change, copy, redistribute
- Derived work must also use the GPL license
- Enables sharing of code
- Simplifies distribution of the program
- Linux is distributed under the GPL license
- See `http://www.gnu.org`

Features

- 2D or 3D
- Automatic assembling
- Triangles or tetrahedrons
- Linear elements
- Algebraic solvers: LU, GMRES, CG, preconditioners

Everyone loves a screenshot



Examples

- *Start movie 1 (driven cavity, solution)*
- *Start movie 2 (driven cavity, dual)*
- *Start movie 3 (driven cavity, dual)*
- *Start movie 4 (bluff body, solution)*
- *Start movie 5 (bluff body, dual)*
- *Start movie 6 (jet, solution)*
- *Start movie 7 (transition to turbulence)*

Input / output

Input / output

- OpenDX: free open-source visualisation program based on IBM:s *Visualization Data Explorer*.
- MATLAB: commercial software (2000 Euros)
- GiD: commercial software (570 Euros)

Note: Input / output has been redesigned in the 0.3.x versions of DOLFIN and support has not yet been added for OpenDX and GiD.

GiD / MATLAB

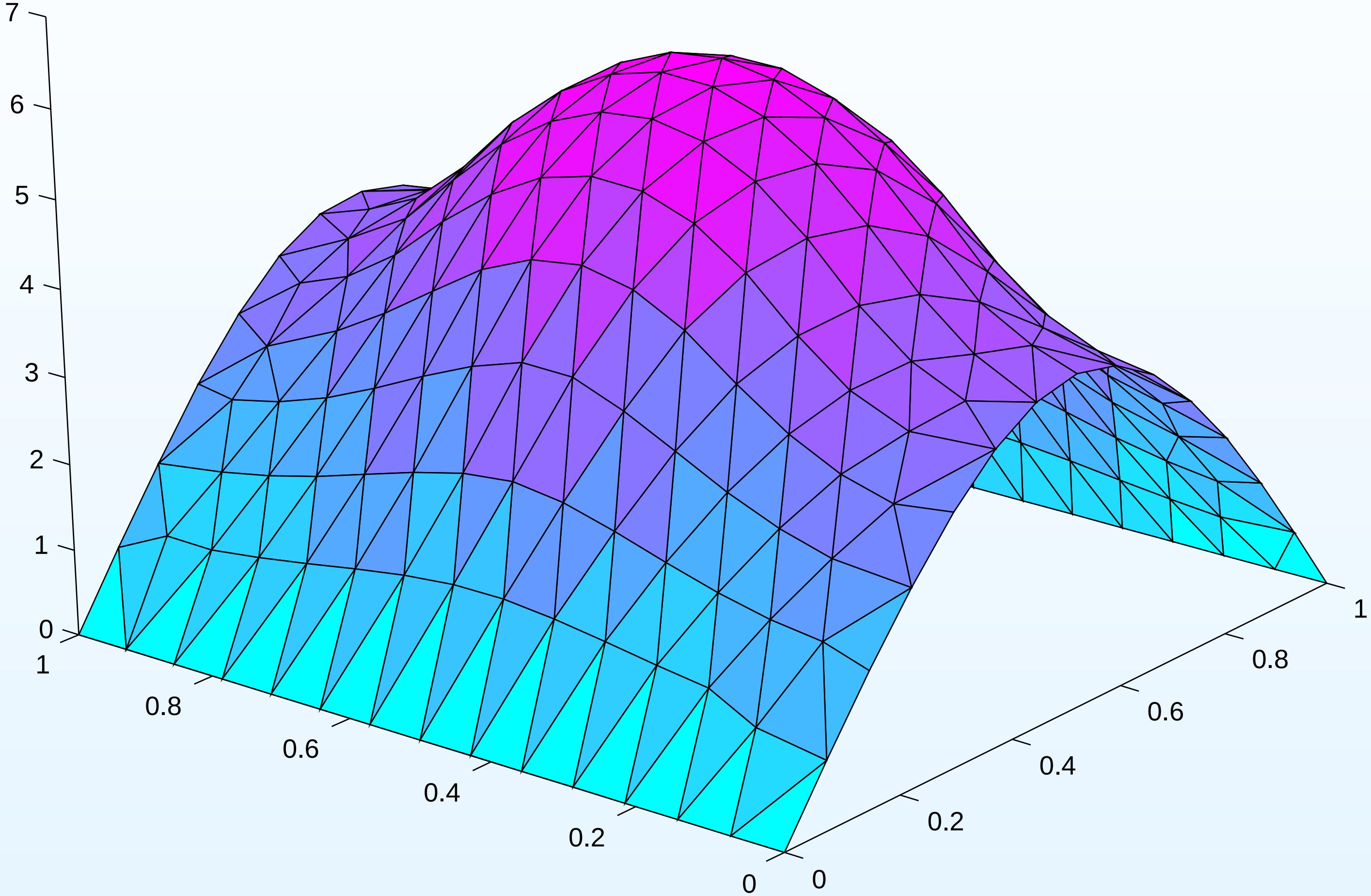
Poisson's equation:

$$-\Delta u(x) = f(x), \quad x \in \Omega, \tag{1}$$

on the unit square $\Omega = (0,1) \times (0,1)$ with the source term f localised to the middle of the domain.

Grid generation with **GiD** and visualisation using the `pdesurf` command in **MATLAB**.

GiD / MATLAB



MATLAB / GiD

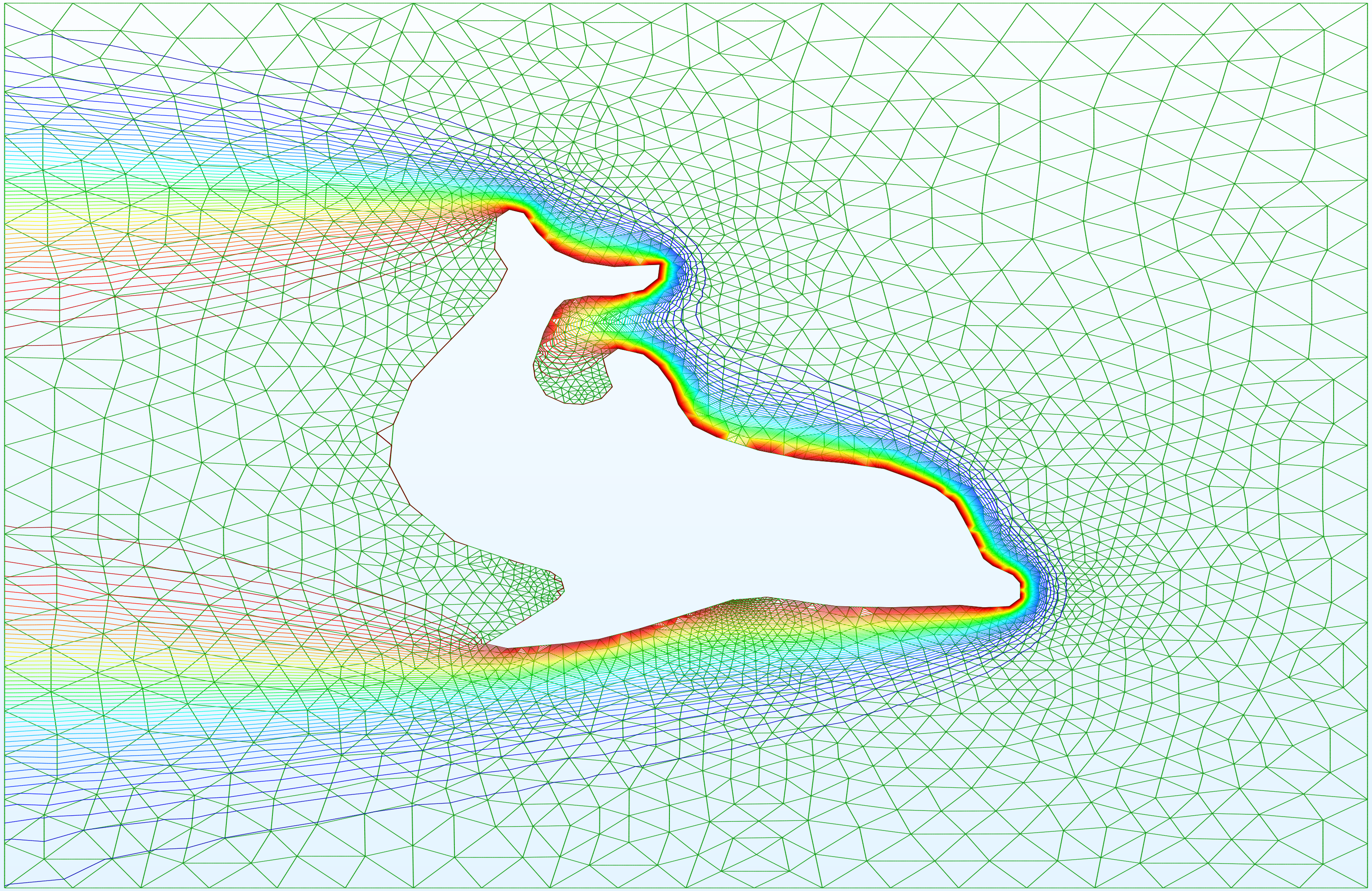
Convection–diffusion:

$$\dot{u} + b \cdot \nabla u - \nabla \cdot (\epsilon \nabla u) = f, \qquad (2)$$

with $b = (-10, 0)$, $f = 0$ and $\epsilon = 0.1$ around a hot dolphin.

Grid generation with **MATLAB** and visualisation using *contour lines* in **GiD**.

MATLAB / GiD



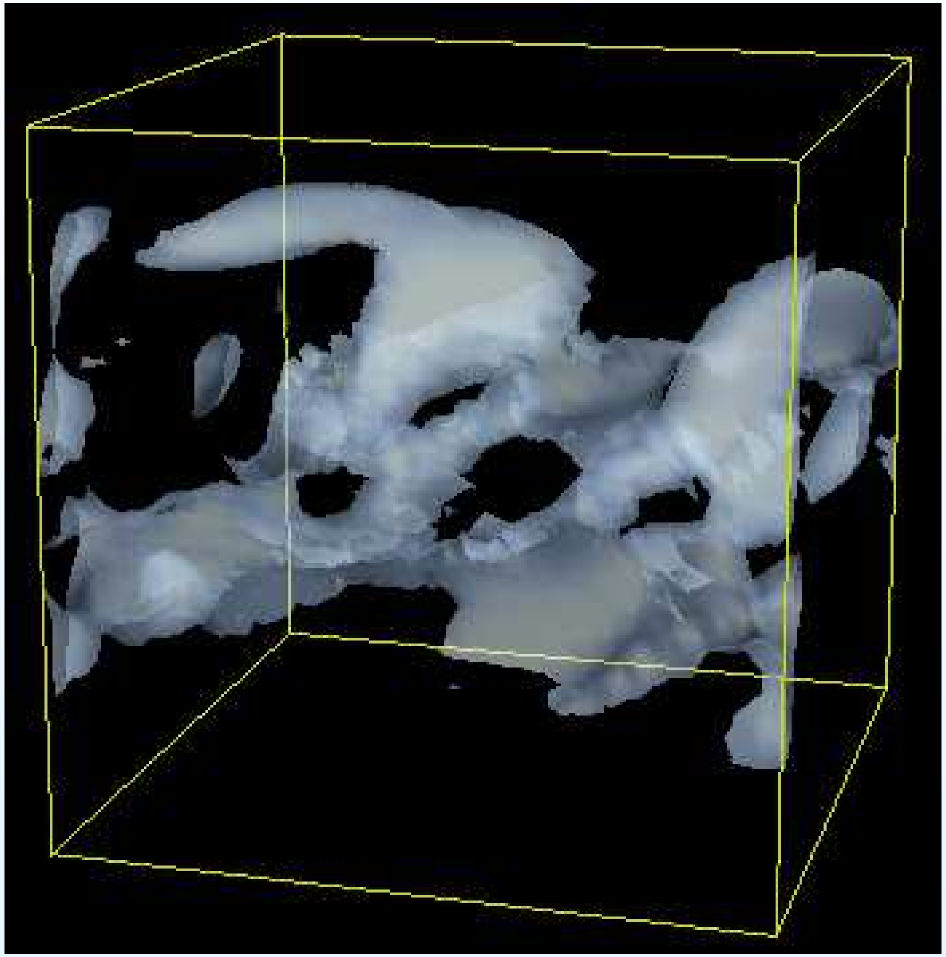
OpenDX

Incompressible Navier–Stokes:

$$\begin{aligned} \dot{u} + u \cdot \nabla u - \nu \Delta u + \nabla p &= f, \\ \nabla \cdot u &= 0. \end{aligned} \tag{3}$$

Visualisation in **OpenDX** of the isosurface for the velocity in a computation of transition to turbulence in shear flow on a mesh consisting of 1,600,000 tetrahedral elements.

OpenDX



A very quick guide to C++

C++

- Invented by Bjarne Stroupstrup at AT&T Bell Laboratories in the early 1980's
- Extends the C programming language to provide support for object-oriented programming
- Widely used
- Standardised by ANSI
- Fundamental concept: *class*

Hello world in C++

```
#include <iostream>

using namespace std;

int main()
{
    cout << ‘‘Hello world!’’ << endl;
    return 0;
}
```

Hello world in C++

```
#include <iostream>

using namespace std;

int main()
{
    int n = 10;
    for (int i = 0; i < n; i++)
        cout << ‘‘Hello world!’’ << endl;
    return 0;
}
```

A basic C++ vocabulary

- Fundamental data types:

```
char, int, float, bool
```

- Conditions and loops:

```
if, else, switch, case,
for, while, break, continue
```

- Classes:

```
class, public, private, protected
```

- General:

```
#include, namespace, new, delete
```

Definition of a variable

- Note how in the Hello World-program each variable is defined as

```
Type name;
```

- A variable must be introduced before it is used.
- A variable can be defined almost anywhere in the code.

Declaration of a class

```
class Vector {
public:
    Vector(int n);        // Constructor
    ~Vector();            // Destructor

    void resize(int n); // A function
    int  size();          // Another function
private:
    int n;                // Size of the vector
    double* values;       // The values
};
```

Using the `Vector` class

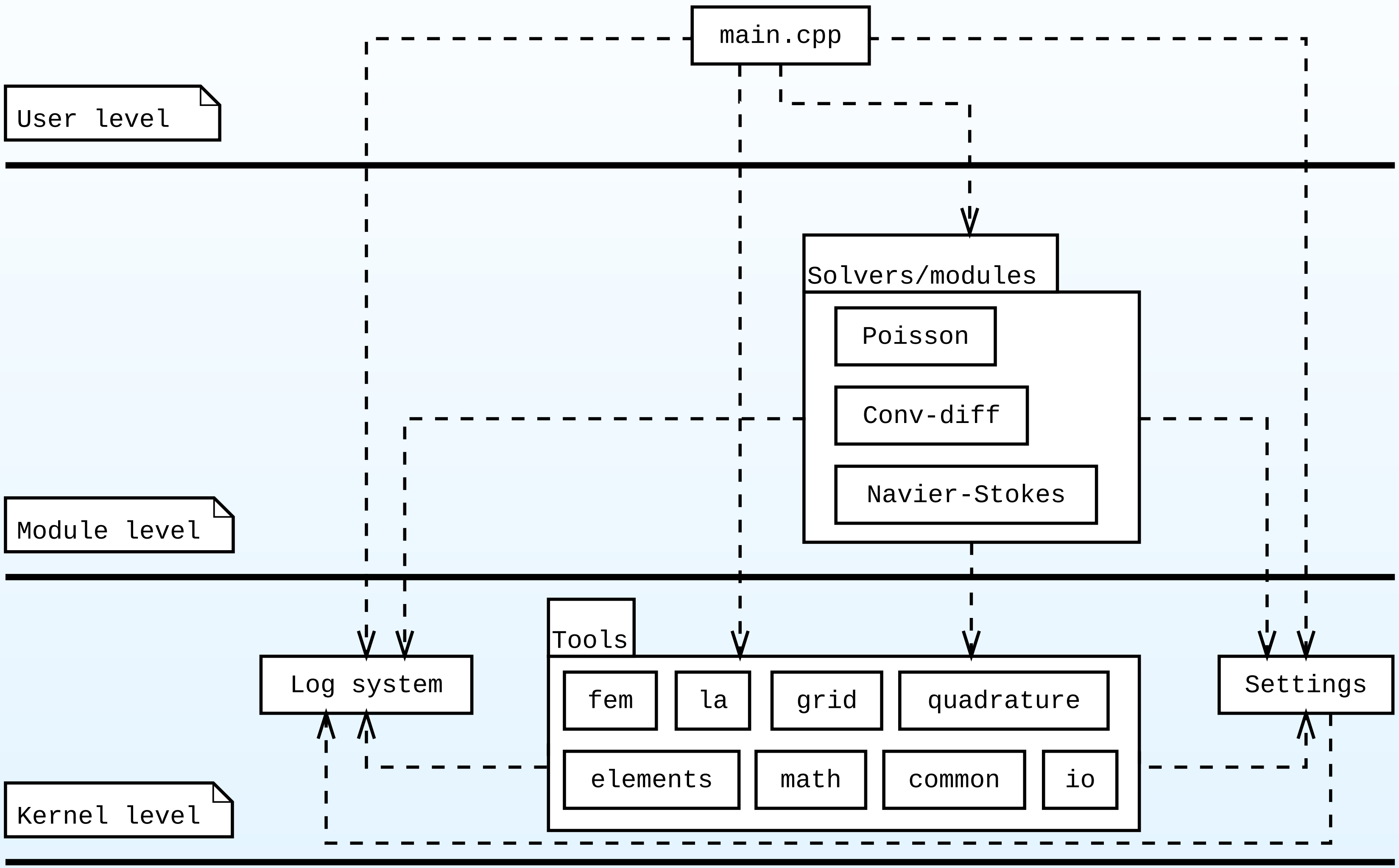
```
Vector x(10);
Vector y(10);

for (int i = 0; i < 10; i++)
    x(i) = (double) i*i;

x *= 5.0; // x <- 5x
y += x;   // y <- y + x
```

Development with DOLFIN

Code structure



Three levels

- Simple C/C++ interface for the *user* who just wants to solve an equation with specified geometry and boundary conditions.
- New algorithms are added at *module level* by the developer or advanced user.
- Core features are added at *kernel level*.

Solving Poisson’s equation

```
int main()
{
  Grid grid("grid.xml.gz");
  Problem poisson("poisson", grid);

  poisson.set("source", f);
  poisson.set("boundary condition", mybc);

  poisson.solve();

  return 0;
}
```

Implementing a solver

```
void PoissonSolver::solve()
{
  Galerkin     fem;
  Matrix       A;
  Vector       x, b;
  Function     u(grid, x);
  Function     f(grid, "source");
  Poisson      poisson(f);
  KrylovSolver solver;
  File         file("poisson.m");

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

  u.rename("u", "temperature");
  file << u;
}
```

Automatic assembling

```
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;
  }
  ...
};
```

Automatic assembling

```
class ConvDiff : public PDE {
  ...
  real lhs(const ShapeFunction& u, const ShapeFunction& v)
  {
    return (u*v + k*((b,grad(u))*v + a*(grad(u),grad(v))))*dK;
  }

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

Grid management

Basic concepts:

- `Grid`
- `Node`, `Cell`, `Edge`, `Face`
- `Boundary`
- `GridHierarchy`
- `NodeIterator`
  `CellIterator`
  `EdgeIterator`
  `FaceIterator`

Grid management

Reading and writing grids:

```
File file(‘‘grid.xml’’);
Grid grid;
file >> grid; // Read grid from file
file << grid; // Save grid to file
```

Grid management

Iteration over a grid:

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

Linear algebra

Basic concepts:

- `Vector`
- `Matrix` (sparse, dense or generic)
- `KrylovSolver`
- `DirectSolver`

Linear algebra

Using the linear algebra:

```
int N = 100;
Matrix A(N,N);
Vector x(N);
Vector b(N);
b = 1.0;
for (int i = 0; i < N; i++) {
  A(i,i) = 2.0;
  if ( i > 0 )
    A(i,i-1) = -1.0;
  if ( i < (N-1) )
    A(i,i+1) = 1.0;
}
A.solve(x,b);
```

Summary of features

Summary of features

Implemented features:

- Automatic assembling
- Linear elements in 2D and 3D
- Basic grid management
- Basic linear algebra
- Solvers for Poisson and convection–diffusion
- Log system
- Parameter management

Summary of features

In preparation:

- Adaptive grid refinement (Hoffman/Logg)
- Multi-adaptive ODE-solver (Jansson/Logg)
- Improved preconditioners (Hoffman/Svensson)
- Improved linear algebra (Hoffman/Logg)

Summary of features

Wishlist / help wanted:

- Multi-grid (Målqvist/Svensson?)
- Implementation of boundary conditions
- Eigenvalue solvers
- Higher-order elements (Svensson?)
- Parallelization
- Documentation
- New solvers / modules (everyone invited!)
- Testing, bug fixes (everyone invited!)

Web page

- `www.phi.chalmers.se/dolfin`

Detailed documentation of the API for DOLFIN is automatically generated every night and is available from the web page.