

# Chapter 1

## Welcome to the homepage of oomph-lib

oomph-lib is an object-oriented, open-source finite-element library for the simulation of multi- (and single-)physics problems, developed and maintained by [Matthias Heil](#) and [Andrew Hazel](#) of the [School of Mathematics](#) at [The University of Manchester](#).

The latest version of the library is revision 0.90 which was released on August 3, 2009. See the [change log page](#) for an overview of the changes made since the previous release.

### 1.1 All you ever wanted to know about oomph-lib

- [What it is \(and what it is not\)?](#)
- [Features](#)
- [How much does it cost? Nothing!](#)
- [How do I get it? What do I need? How do I install it?](#)
- [How do I visualise the results?](#)
- [Who develops/maintains oomph-lib?](#)
- [Contact/Further information](#)
- [Acknowledgements](#)

### 1.2 What it is (and what it is not)?

oomph-lib **is**:

- an object-oriented, open-source finite-element library for the simulation of multiphysics problems.

oomph-lib **it is not**:

- a GUI-based, mouse-driven "package".

You must write your own C++ driver codes to specify your problem, using oomph-lib's high-level objects. [If your C++ is a bit rusty (or non-existent) have a look at Andrew Hazel's [C++ course](#) for a quick introduction.] Once the problem is formulated, the `main` function can be as simple as this:

```
main()
{
    // Create the problem object
    MyProblem problem;

    // Solve the problem, using oomph-lib's default Newton solver
    problem.newton_solve();
}
```

oomph-lib provides the infrastructure for the problem formulation and solution. The library contains a large number of elements, meshes, timesteppers, solvers, etc. which can be combined to solve any problem.

The library is supplied with extensive on-line documentation which includes:

- A [top-down introduction](#): This provides a relatively brief, constructive introduction to the finite element method, and explains its object-oriented implementation in oomph-lib.
- A [bottom-up discussion](#) of oomph-lib's overall data structure, including a complete, cross-referenced, html-based index of all oomph-lib classes.
- The [\(Not-So-\)Quick Guide](#) provides a "quick" introduction on how to create new instances of oomph-lib's fundamental objects: Problems, Meshes, and Elements.
- A [large number of example problems](#)
  - The example problems are arranged in order of increasing complexity and can be used as chapters in a self-study course. Comments and exercises are provided to encourage further exploration of oomph-lib's capabilities.
  - The individual examples are as self-contained as possible and can be used as quick guides to particular features of oomph-lib. ("How do I solve time-dependent problems?"; "How do I enable spatial adaptivity?"; "How do I write my own meshes?"; etc.)
  - Finally, the examples codes should be regarded as templates for your own driver codes: Examine the [list of examples](#) and try to find one that resembles the problem that you wish to solve. For most "classical" problems (e.g. the solution of the Navier-Stokes equations with standard boundary conditions), very few modifications are required (e.g. adjusting the mesh to a different domain, changing the boundary and/or initial conditions, etc).  
If you wish to solve a non-standard problem, or one for which oomph-lib does not (yet) provide elements or meshes, it is easy to write your own, following the detailed instructions provided in the documentation.

## 1.3 Features

- Large number of fully-developed and carefully-validated element types for the solution of:
  - The Poisson equation
  - The unsteady heat equation
  - The linear wave equation
  - The advection-diffusion equation
  - The Navier-Stokes equations, incl. free-surfaces, in Cartesian and axisymmetric co-ordinates.
  - The equations of large-displacement solid mechanics, based on the principle of virtual displacements in cartesian and axisymmetric geometries; with arbitrary constitutive equations.
  - Shell and beam elements, based on a geometrically non-linear formulation.

[**Note:** Most elements are implemented in a dimension-independent formulation]

- It is easy to formulate new elements for problems that are not included in the above list.
- It is easy to combine any of the above elements to "multi-physics" elements, allowing e.g.
  - the solution of fluid-structure interaction problems
  - the simulation of unsteady heat conduction in an elastic body that undergoes large displacements
  - free-surface problems
  - etc.
- Parallel processing capabilities.
- Full spatial and temporal adaptivity. All elements listed above support quad-tree/octree-based mesh refinement procedures and adaptive timestepping.
- A wide range of meshes; new meshes can easily be added to the library. `oomph-lib` can also use meshes that are generated by third-party (commercial) mesh generators.
- The ability to solve problems in domains with time-dependent, curvilinear boundaries.
- `oomph-lib` treats all problems as non-linear problems and uses Newton's method as the default non-linear solver. Within this framework, linear problems are special cases for which the Newton iteration converges in one step. The linear systems arising during the Newton iteration can be solved with a variety of direct and iterative solvers. Continuation methods (Keller's arclength method and displacement-control for solid mechanics problems) are implemented.
- A large number of **fully documented example codes**. All examples include an introduction to the mathematical/physical problem, a detailed discussion of the driver code, a discussion of the results (incl. comparisons against analytical solutions where appropriate) and exercises and comments.
- The library has extensive self-test facilities: Every sufficiently complex class has a self-test function that performs various sanity checks – useful during code-development. The library can be compiled with a `PARANOID` flag switched on or off. If the flag is set during compilation, the code performs a large number of self-tests during its execution. If things go wrong, code execution terminates gracefully and diagnostic output is generated to help pinpoint the problem. Of course, the tests introduce an additional run-time overhead, so can be switched off in production runs. During development, range checking on all array-based objects, (vectors, matrices, tensors, etc) can be enabled separately by using the `RANGE_CHECKING` flag, but introduces considerable run-time overhead.

## 1.4 How much does it cost? Nothing!

- `oomph-lib` is freely available under the **gnu public licence**. Help yourself!
- We welcome feedback and will try our best to help you if you get stuck. However, **please** read the documentation and the FAQ first! We are busy academics and do not have the resources to operate a commercial-style "helpline".
- "Free" under the **gnu public licence** means "free for all" – this includes commercial users. The same applies (within our limited resources!) to requests for help. However, if you are a commercial user and are interested in dedicated assistance for a specific project, let us know. We'll be happy to discuss the possibility of consultancy arrangements.

## 1.5 How do I get it? What do I need? How do I install it?

### 1.5.1 How do I get it?

We provide [gzipped tar files](#) of the entire library (big!), and (much smaller) partial distributions in which, e.g. the online documentation and/or the demo/self-test codes and the associated validation data are excluded.

We also provide read-only access to the [subversion](#) repository of the oomph-lib project which is hosted at [oomph-lib.maths.man.ac.uk](http://oomph-lib.maths.man.ac.uk). If [subversion](#) is installed on your computer you can obtain the latest snapshot of the project by issuing the command

```
svn checkout svn://oomph-lib.maths.man.ac.uk/trunk
```

See [our subversion webpage](#) for further details.

### 1.5.2 What do I need?

oomph-lib has been/is being developed in a linux environment, using standard ANSI C++. In order to minimise the dependence on external libraries we include "frozen" versions of certain external libraries ([METIS](#), [SuperLU](#), ...) with our distribution. The idea is that you should only have to unpack, build and install one distribution to produce the fully compiled library, the html-based documentation and working example codes.

For this purpose you **must have**:

- A computer with a linux (or unix) operating system. [**Note for windows users:** oomph-lib can be installed in a windows environment, using [cygwin](#).]
- Compilers for C++, C and Fortran77. The open-source [GNU gcc compiler suite](#) version 3.2.3 or later is fine; the library also compiles cleanly with the Intel compilers ([icpc](#), [icc](#) and [ifc](#)). [**Note:** The Fortran77 and C compilers are only required to compile some external libraries – oomph-lib itself is written entirely in C++.]

The following programs are helpful but not essential:

- GNU's [autoconf](#), [automake](#) and [libtool](#) are required if you wish to add additional features to your local copy of oomph-lib; see [the oomph-lib download page](#) for more details.
- [python](#) is needed to analyse the results of the extensive self-tests that may be performed at the end of the installation to verify that the build process was completed successfully. If python is not available, the self-test will compile and run all test codes but it will not be possible to verify the correctness of the results.
- [doxygen](#) is needed to build a local copy of oomph-lib's extensive online documentation. If doxygen is not installed on your system you can consult the latest version of the online documentation on [the oomph-lib homepage](#).

### 1.5.3 How do I install it?

The oomph-lib distribution is built under [autoconf](#) / [automake](#) control, making the installation completely straightforward. We provide two main installation mechanisms:

1. `oomph-lib` can be built/installed/tested with GNU's standard `configure`; `make`; `make install`; `make check` procedure.
2. We also provide a customised build script, `autogen.sh`, that guides the "non-expert" user through the installation. The script facilitates the specification of various build options, such as optimisation levels etc.

Subject to the minimum requirements, listed above, you should be able to install and use our libraries "anywhere". However, as anybody who has ever ported any code to a different platform knows, things are rarely that simple – even with `autoconf`, `automake` and `libtool`... Having said that, we have successfully installed and tested our installation on the following platforms:

- Redhat Linux, using the gcc compiler suite (versions 3.2.3, 4.1.2 and 4.3.3). The tests included the MPI part of the library, which we compiled with `lam`.
- Gentoo Linux, using the gcc compiler suite (versions 3.3.4 and 4.1.2) and the intel compilers (`icpc`, `ifc` and `icc` all at version 10).
- Ubuntu (version 9.04) with gcc compilers (version 4.3.3). The tests included the MPI part of the library, which we compiled with `lam`.
- Apple Mac OSX 10.4.11 (Tiger) with gcc compilers (version 4.3.3)
- Apple Mac OSX 10.5.8 (Leopard) with gcc compilers (version 4.3.3)
- cygwin running under Windows XP, using the gcc compiler suite (version 3.4.4).
- cygwin under Windows Vista Business using the gcc compiler suite (version 3.4.4).
- cygwin under Windows 7 Ultimate (evaluation copy, build 7100) using the gcc compiler suite (version 3.4.4).

We welcome feedback on any problems that you encounter during your attempts to install `oomph-lib` on your machine. However, **please** check the [FAQ](#) and the instructions on the [download page](#) before you contact us!

## 1.6 How do I visualise the results?

An important question! As mentioned above, `oomph-lib` is not a "GUI package" and does not provide any built-in visualisation tools. `oomph-lib`'s post-processing routines produce ASCII data in a format that is suitable for post-processing with `tecplot` – a powerful and easy-to-use commercial plotting package. [The plots and animations shown in the [examples](#) were all produced with `tecplot`.] `gnuplot` can also display the data (in more elementary form, obviously). The trick is to specify the `using` option in `gnuplot`'s plot commands – in this mode `gnuplot` ignores `tecplot`'s "ZONE" commands. **Angelo Simone** wrote python scripts that convert `oomph-lib`'s output to the vtU format that can be read by `paraview`; see [the paraview tutorial](#) for details.

## 1.7 Who develops/maintains oomph-lib?

The `oomph-lib` "architects" are (in no particular order)



## Andrew Hazel



## Matthias Heil

...assisted by former/current project/MSc/PhD students and collaborators who made (or are still making) significant contributions to the development of the library (listed in reverse chronological order):

- **Benjamin Metz** works on adaptivity and solution transfer for unstructured meshes.
- **Amine Massit** works on outflow boundary conditions for Navier-Stokes problems and physiological FSI problems based on meshes generated by [vmtk](#).
- **Patrick Hurley** works on free surface Navier-Stokes problems.
- **Andy Gait** works on parallelisation, in particular the problem distribution and the subsequent distributed mesh adaptation.
- **Angelo Simone** wrote python scripts that convert oomph-lib's output to the vtu format that can be read by [paraview](#); see [the paraview tutorial](#) for details.
- **Floraine Cordier** developed the driver codes and tutorials for the [flow past the elastic leaflet](#) and [Turek & Hron's FSI benchmark](#). In the process she significantly extended oomph-lib's FSI capabilities.
- **Stefan Kollmannsberger** and his students Iason Papaioannou and Orkun Oezkan Doenmez developed early versions of the code for [Turek & Hron's FSI benchmark](#) and its [non-FSI counterpart](#).
- **Cedric Ody** developed the YoungLaplace elements and their refineable counterparts to study capillary statics problems.
- **Alice Gaertig** developed interfaces to the third-party mesh generators [Triangle](#), [TetGen](#), [Geompack++](#), and [CQMesh](#).
- **Claire Blancon** developed the demo drivers for the collapsible channel problem (with and without fluid-structure interaction).
- **Nick Chapman** worked on the implementation of triangular and tet-elements.
- **Chris Gold** implemented explicit timestepping schemes.
- **Richard Muddle** works on the block preconditioning techniques for the biharmonic (and many other) equations, and parallel solvers.
- **Glyn Rees** works on iterative linear solvers and multigrid
- **Alberto de Lozar** worked on 3D free-surface Navier-Stokes problems.

- **Jonathan Boyle** developed the initial interfaces to third-party iterative solvers and is now involved in the further parallelisation of the code and the implementation and application of block-preconditioning techniques for Navier-Stokes and fluid-structure interaction problems.
- **Renaud Schleck** completed the octree-based mesh refinement procedures and wrote the MPI-based parallel assembly routines and the interfaces to SuperLU\_dist.
- **Sharaf Al-Sharif** provided the initial implementation of nodal spectral elements.
- **Daniel Meyer** used oomph-lib to study a variety of axisymmetric Navier-Stokes problems, with and without free surfaces, and developed drafts for many of our tutorials.
- **Alexandre Klimowicz** worked on block-preconditioning methods.
- **Jean-Michel Lenoir** implemented the first part of the octree-based 3D mesh refinement procedures.
- **Gemma Barson** provided the initial implementation for the 2D Delaunay mesh generation procedures.

We're always looking for more help! Get in touch if you're interested in joining the team, and check out the [ToDo page](#) for capabilities/features that are planned for future releases.

## 1.8 Contact/Further information

You can contact the developers at

oomph-lib AT maths DOT man DOT ac DOT uk

If you wish to be kept up-to-date about revisions, bug fixes and new releases, join our (very low volume) mailing list.

## 1.9 Acknowledgements

We wish to acknowledge the direct and indirect support from the following people and organisations:

- The [EPSRC](#) (the UK Engineering and Physical Sciences Research Council) for their financial support.
- [MIMS](#), the [Manchester Institute for Mathematical Sciences](#) and the [School of Mathematics at The University of Manchester](#) for their financial support.
- Dimitry van Heesch for his brilliant [doxygen](#) package which we used to produce the documentation.
- [Xiaoye Sherry Li](#) and colleagues for developing the sparse direct solver [SuperLU](#) – oomph-lib's default linear solver.
- [Chris Paul](#), Head of Computing at [Manchester's School of Mathematics](#) for keeping the machines running!
- The [Manchester Centre for Novel Computing](#), for giving us access to their Silicon Graphics Origin 2000, for the early development of oomph-lib's MPI routines.
- [Daoqi Yang](#) for his brilliant book [C++ and Object-Oriented Numeric Computing for Scientists and Engineers](#).

## 1.10 PDF file

A [pdf version](#) of this document is available.