Chapter 1

Frequently Asked Questions

Here are some frequently asked questions (with assorted frequently given answers...). Please check these before contacting us with any problems.

1.1 Installation

1.1.1 How do I specify different levels of optimisation and/or non-standard compiler options?

1. **Compiler flags/options for the compilation of the library**

   By default, `oomph-lib's` installation is performed with the `gcc compiler suite`, using full optimisation and warning (`-O6 -Wall`).

   If you install the library with the `autogen.sh` script you are given the option to overwrite the default compilation flags, either by specifying alternative flags on the command line, or by recycling any of the previously used combinations of flags, stored in the directory `config/configure_options`. The files in this directory also contain more details on available flags and options.

   If you prefer the standard linux `configure/make/make install` installation procedure, you should already know how to specify flags at the configure stage...

2. **Compiler flags/options for the compilation of individual (driver) codes**

   Within the `autotools` framework, additional flags for the compilation of individual (driver) codes may be specified by setting the appropriate automake variables in the `Makefile.am`. For instance, to use the compiler flag `-DUSE_TAYLOR_HOOD` during the compilation of the executable `test_code`, add

   ```
   test_code_CXXFLAGS = -DUSE_TAYLOR_HOOD
   ```

   to the appropriate `Makefile.am`. The C++ section of the `automake manual` provides more detail on other `automake` variables.

   If you link against `oomph-lib` from outside the `autotools` framework, you can, of course, use whatever technique you prefer to customise the compilation of your driver code.

1.1.2 The compilation fails when I use my xxx [non gcc] compiler

`oomph-lib` is developed in a GNU Linux environment, using the `gcc compiler suite`. We believe (!) the source code complies with the `C++ standard` and it compiles cleanly under the `gcc compilers` that we have access to (version 3.2.3 and later). Some warning messages tend to be issued during the compilation of the third-party libraries distributed with `oomph-lib`, but these are not our responsibility!

We occasionally compile (or get other people to compile) the library under other compilers and we generally try to rectify any problems that are flagged up in the process. If you encounter any problems (errors or warning messages) while compiling the library with your own (non-gcc) compiler we would `like to hear from you`, especially if you suggest concrete bug fixes.

There are problems with version 7 of the intel compiler suite but the library compiles cleanly under version 9. We suggest you upgrade to that if you wish to use the intel compilers.

1.1.3 The build process fails under cygwin

Here are a few things to check:

- Make sure the gcc compiler suite is sufficiently up-to-date (we recommend version 3.2.3 or later).

- The build process will fail if you have (foolishly!) chosen a windows username that contains spaces. The build script (executed under linux) makes frequent references to the (absolute!) path to the `oomph-lib` installation directory. Linux does not allow spaces in directory names!

1.1.4 The build process fails under Apple's Darwin (OSX) or other BSD Linux distributions

Here are some general guidelines for installing under OSX, contributed by `Rich Hewitt`:

1. Make sure you have installed the development tools from the OSX install DVDs. If your install media is fairly old, you may wish to look for an updated online version at:

   `http://developer.apple.com/tools/xcode/`

   You do not need to use the Xcode IDE, but we do need the associated development kit tools (in particular GCC!)

2. Install a FORTRAN compiler. Apple does not ship a FORTRAN version, so the easiest way to do this is to install the relevant pre-packaged GNU FORTRAN from here:

   `http://hpc.sourceforge.net/`

   (I would recommend sticking with the version of GCC4 provided by Apple.)

3. Make sure the FORTRAN compiler ('gfortran' in the packages pointed to above) is in the installing user's path.

4. Follow the `oomph-lib install procedure` as normal.

5. For associated Unix-goodness (doxygen, gnuplot, subversion etc), I would recommend using a ports mechanism to install any additional unix tools painlessly:

   `https://www.macports.org/`

   For example: "sudo port install gnuplot" etc.

1.1.5 The linking self-test fails under Apple's OSX

As discussed in the installation instructions, Darwin (the BSD-derived UNIX core of Apple's OSX operating system) requires a slight change to the default procedure for linking against oomph-lib from outside its autotools framework. As a result, the linking self-test tends to fail on machines with this operating system. Instructions on how to fix this problem are provided in installation instructions.

1.1.6 Some of the self tests fail

Tolerance of comparison between floating point numbers

The full oomph-lib distribution contains a large number of demo codes in the sub-directory demo_drivers. Primarily, these codes serve as demo codes for the oomph-lib documentation (contained in the sub-directory doc) but they are also used during the library's self-test procedure which checks that oomph-lib was installed correctly. The self-tests can either be initiated at the end of the autogen.sh - based installation or by typing

```
make check -k
```

in oomph-lib's top-level directory. The self-test builds and runs all demo codes and compares their results against the reference data stored in the validata sub-directories. The comparison is performed with the python script bin/fpdiff.py which tolerates small (machine- and compiler-dependent) roundoff errors and suppresses the comparison of numbers whose absolute value falls below some threshold (to avoid the comparison of numerical zeroes).

If any of the self-test fail, you should first check the output in the file validation.log to assess if the differences between the computed data and the reference data are significant. If the discrepancy appears to be due to larger-than-anticipated round-off errors (you'll have to judge this yourself!), modify the validate.sh script to specify a larger relative tolerance and/or a larger value for the threshold below which fpdiff.py regards numbers as numerical zeroes and excludes them from the comparison. [Type

```
bin/fpdiff.py
```

in oomph-lib's top-level directory for instructions on how to use the script].

The self-test fails even though the output files produced by the code are correct

This is an odd error that is usually caused by the use of wildcards in the validations scripts, or the comparison of data that is stored in certain STL containers. Typically, the self-test is performed by the validate.sh shell script, which runs the executable and concatenates selected output files to a single file whose contents are compared against the reference file in the validata directory.

While it is tempting to write

```
cat RESLT* /soln0.dat > results_file.dat
```

it is important to realise that the order in which the files are concatenated is machine- and/or operating-system dependent. If the above command is run in a directory with the following structure

```
|-- RESLT
|   |-- soln0.dat
|   `-- trace.dat
|-- RESLT_elastic
|   |-- soln0.dat
|   `-- trace.dat
```

some operating systems will expand the command to

```
cat RESLT/soln0.dat RESLT_elastic/soln0.dat > results_file.dat
```

while others will execute

```
cat RESLT_elastic/soln0.dat RESLT/soln0.dat > results_file.dat
```

In this case the self-test will report a failure, even though the solution files are correct. The `validate.sh` scripts should therefore not contain any wildcards.

Similar problems can arise if the validation data includes data that is stored in certain STL containers such as sets. The order in which items are stored in such containers may vary from machine to machine and from compiler to compiler. If such data is to be included in a self-test the data should be sorted first, based on a user-controllable sorting criterion.

1.1.7 How do I compile/run the demo driver codes?

Assume you have followed our advice to explore `oomph-lib` by playing with representative demo driver codes, and have ended up in a directory that contains the following files:

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ ls -l
total 68
-rw-rw-r-- 1 mheil mheil 21604 2012-12-31 11:10 Makefile
-rw-rw-r-- 1 mheil mheil   409 2012-12-08 13:34 Makefile.am
-rw-rw-r-- 1 mheil mheil 19335 2012-12-31 11:09 Makefile.in
-rw-rw-r-- 1 mheil mheil  9638 2012-12-08 13:34 one_d_poisson.cc
drwxrwxr-x 3 mheil mheil  4096 2012-12-08 13:34 validata
-rwxrwxr-x 1 mheil mheil  2402 2012-12-08 13:34 validate.sh
```

The temptation is to compile the driver code `one_d_poisson.cc` with your favourite C++ compiler, e.g. `g++`:

```
g++  one_d_poisson.cc
```

but this does **not** work:

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ g++ one_d_poisson.cc
one_d_poisson.cc:31:21: fatal error: generic.h: No such file or directory
compilation terminated.
```

Why? If you look into the code, you notice that it #includes `oomph-lib's generic.h` header file, and `g++` obviously doesn't know where to find this. You can, in principle, find out where it (and the associated library files themselves!) live but this is rather against the spirit of the `automake` machinery. What you should do instead is to use `make`:

```
make one_d_poisson
```

Note the omitted postfix! The argument to `make` is the name of the target, i.e. the executable that you wish to build – which source code to compile, where to locate the required headers and which libraries to link against is all encoded in the `Makefile` (which has itself been created by `automake` (during the installation of the `oomph-lib`) from the `Makefile.am` file in the directory). As you see from the on-screen output, there is a lot of information that needs to be passed to the compiler/linker:

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ make one_d_poisson
mpic++ -DHAVE_CONFIG_H -I. -I../../..  -I/home/mheil/local/hypre_default_installation_mpi/include -
      DOOMPH_HAS_HYPRE -I/home/mheil/local/trilinos_default_installation_mpi/include -DOOMPH_HAS_TRILINOS -I/home/mheil/
      local/mumps_and_scalapack_default_installation/include -DOOMPH_HAS_MUMPS -DOOMPH_HAS_STACKTRACE -
      DOOMPH_HAS_MPI -DOOMPH_HAS_TRIANGLE_LIB -DOOMPH_HAS_TETGEN_LIB -DUSING_OOMPH_SUPERLU -DUSING_OOMPH_SUPERLU_DIST -I/home/
      mheil/version339/build/include  -DgFortran -O3 -Wall -MT one_d_poisson.o -MD -MP -MF .deps/one_d_poisson.Tpo
      -c -o one_d_poisson.o one_d_poisson.cc
mv -f .deps/one_d_poisson.Tpo .deps/one_d_poisson.Po
/bin/bash ../../../libtool --tag=CXX   --mode=link mpic++ -DgFortran -O3 -Wall -L/home/mheil/local/
      hypre_default_installation_mpi/lib -L/home/mheil/local/trilinos_default_installation_mpi/lib -L/home/mheil/local/
      mumps_and_scalapack_default_installation/lib  -o one_d_poisson one_d_poisson.o -L/home/mheil/version339/build/
      lib -lpoisson -lgeneric -lHYPRE -lml -lifpack -lamesos -lanasazi -laztecoo -lepetraext -ltriutils -lepetra -
      lteuchos -ldmumps -lmumps_common /home/mheil/local/mumps_and_scalapack_default_installation/lib/libscalapack.
      a /home/mheil/local/mumps_and_scalapack_default_installation/lib/blacs.a /home/mheil/local/
      mumps_and_scalapack_default_installation/lib/blacsF77.a /home/mheil/local/mumps_and_scalapack_default_installation/lib/
      blacs_copy.a /home/mheil/local/mumps_and_scalapack_default_installation/lib/libpord.a -loomph_hsl -loomph_arpack -
      loomph_triangle -loomph_tetgen -loomph_superlu_4.3 -loomph_parmetis_3.1.1 -loomph_superlu_dist_3.0 -
      loomph_metis_from_parmetis_3.1.1 /home/mheil/local/lib/lapack/lapack.a /home/mheil/local/lib/blas/blas.a -L/usr/lib/
      openmpi/lib -L/usr/lib/gcc/x86_64-linux-gnu/4.6.1 -L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../../x86_64-
      linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib
      /x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../.. -lmpi_f77 -lmpi -lopen-
      rte -lopen-pal -ldl -lnsl -lutil -lgfortran -lm -lquadmath -lpthread
libtool: link: mpic++ -DgFortran -O3 -Wall -o one_d_poisson
one_d_poisson.o
-L/home/mheil/local/hypre_default_installation_mpi/lib
-L/home/mheil/local/trilinos_default_installation_mpi/lib
-L/home/mheil/local/mumps_and_scalapack_default_installation/lib
-L/home/mheil/version339/build/lib
/home/mheil/version339/build/lib/libpoisson.a
/home/mheil/version339/build/lib/libgeneric.a -lHYPRE -lml -lifpack
-lamesos -lanasazi -laztecoo -lepetraext -ltriutils -lepetra -lteuchos
-ldmumps -lmumps_common
/home/mheil/local/mumps_and_scalapack_default_installation/lib/libscalapack.a
/home/mheil/local/mumps_and_scalapack_default_installation/lib/blacs.a
/home/mheil/local/mumps_and_scalapack_default_installation/lib/blacsF77.a
/home/mheil/local/mumps_and_scalapack_default_installation/lib/blacs_copy.a
/home/mheil/local/mumps_and_scalapack_default_installation/lib/libpord.a
/home/mheil/version339/build/lib/liboomph_hsl.a
/home/mheil/version339/build/lib/liboomph_arpack.a
/home/mheil/version339/build/lib/liboomph_triangle.a
/home/mheil/version339/build/lib/liboomph_tetgen.a
/usr/lib/openmpi/lib/libmpi_cxx.so
/home/mheil/version339/build/lib/liboomph_superlu_4.3.a
/home/mheil/version339/build/lib/liboomph_parmetis_3.1.1.a
/home/mheil/version339/build/lib/liboomph_superlu_dist_3.0.a
/home/mheil/version339/build/lib/liboomph_metis_from_parmetis_3.1.1.a
/home/mheil/local/lib/lapack/lapack.a
/home/mheil/local/lib/blas/blas.a -L/usr/lib/openmpi/lib
-L/usr/lib/gcc/x86_64-linux-gnu/4.6.1
-L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../../x86_64-linux-gnu
-L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../../../lib
-L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu
-L/usr/lib/../lib
-L/usr/lib/gcc/x86_64-linux-gnu/4.6.1/../../.. /usr/lib/openmpi/lib/libmpi_f77.so
/usr/lib/openmpi/lib/libmpi.so /usr/lib/openmpi/lib/libopen-rte.so
/usr/lib/openmpi/lib/libopen-pal.so -ldl -lnsl -lutil -lgfortran -lm
-lquadmath -lpthread -pthread
```

Looking at the content of the directory again shows that the required executable has now been generated:

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ ls -l
total 9988
-rw-rw-r-- 1 mheil mheil   21604 2012-12-31 11:10 Makefile
-rw-rw-r-- 1 mheil mheil     409 2012-12-08 13:34 Makefile.am
-rw-rw-r-- 1 mheil mheil   19335 2012-12-31 11:09 Makefile.in
-rwxrwxr-x 1 mheil mheil 9608851 2012-12-31 15:14 one_d_poisson
-rw-rw-r-- 1 mheil mheil    9638 2012-12-08 13:34 one_d_poisson.cc
-rw-rw-r-- 1 mheil mheil  548256 2012-12-31 15:14 one_d_poisson.o
drwxrwxr-x 3 mheil mheil    4096 2012-12-08 13:34 validata
-rwxrwxr-x 1 mheil mheil    2402 2012-12-08 13:34 validate.s
```

1.1.8 Why doesn't make make anything? What are the targets called?

If you are familiar with makefiles, you are unlikely to have fallen into the trap discussed above. You will almost certainly have spotted the `Makefile` and tried to build the executable(s) with `make`, but oddly this doesn't work as you (probably) expected:

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ make
make: Nothing to be done for 'all'.
```

The reason for this is that

1. `automake` distinguishes between several types of targets,
2. the demo driver codes in the subdirectories of `demo_drivers` double up as self-tests to assert the correct installation of `oomph-lib` when `make check` is run from the top-level directory.

The latter requires them to be declared (in the `Makefile.am`) in the `check_PROGRAMS` variable. You can either build (and automatically run) all executables in a `demo_driver` directory by issuing the command

```
make check -k
```

or you can inspect the `check_PROGRAMS` variable in the `Makefile.am` to find out which targets have been declared (and can therefore be built by `make`). For instance, in the `one_d_poisson` directory the first few lines of the `Makefile.am` are

```
mheil@biolaptop:~/version339/demo_drivers/poisson/one_d_poisson$ more Makefile.am
[...]
check_PROGRAMS=one_d_poisson
one_d_poisson_SOURCES = one_d_poisson.cc
[...]
```

This declares the target (the executable called `one_d_poisson`) and specifies the source code to be compiled (`one_d_poisson.cc`). The build target needs to specified explicitly:

```
make one_d_poisson
```

We note that the names of the executable and the source code do not have to be as closely related as in this example. While we usually create the name of the executable by dropping the postfix of the source file, this is not guaranteed! Inspect the `Makefile.am` if `make` doesn't give you what you want.

1.1.9 Running the self-tests takes a long time. Is it possible to suppress the execution of the built driver codes and only check if they compile? (Mainly useful for developers).

Issuing `make check` from the top level `oomph-lib` directory descends into all oomph-lib directories that contain demo driver codes, builds these and then runs them using the script `validate.sh` which typically also compares the output against some reference data that is stored in the `validata` sub-directories. This process (which allows for some tolerances in floating point data) assesses that the installation is working properly, but the entire procedure obviously takes a long time. As a developer, you may want to check (more) quickly if the changes you've made to any source code (in `src`) still compile without running any of the self-tests. It is easy to suppress the execution of the self tests because the `validate.sh` script is, in fact, run through the wrapper script

```
bin/wrapped_validate.sh
```

which, by default, runs `validate.sh` through `time`, allowing us to report the run times. However, you can easily edit this file and make it do other things. For instance, if you put `exit` onto the first line it won't do anything at all. Success! If you then issue

```
make check
```

(without the "-k"!) from the top level, the build machinery will descend through all the demo driver directories, build the executables and stop when it encounters the first error. If you want to continue with the compilation after encountering failures do

```
make -k check
```

You can assess all the failures at the same time by searching backwards through the on-screen output.

1.1.10 I only want to run the self-tests for demo driver codes that contain a certain string (typically a class name; again mainly useful for developers)

The script

```
bin/run_selected_self_tests_based_on_string.bash
```

will search through all demo driver directories and run the self tests in all directories where the source code of a demo driver contains a certain string. You can specify the string and the search directory in the script or via the command line. Run

```
bin/run_selected_self_tests_based_on_string.bash --help
```

for details.

1.1.11 The oomph-lib distribution includes some third-party libraries. How do I get the code to link against optimised local versions of these libraries that are already installed on my machine?

To facilitate the installation, the oomph-lib distribution includes the relevant parts of certain third-party libraries, such as SuperLU and BLAS. These "external libraries" are built and installed along with oomph-lib's own sub-libraries. To distinguish them from any already existing local versions of these libraries, their names are pre-fixed with the string oomph_.

All demo codes are automatically linked against these libraries. This is achieved by defining the variable EXTERNAL_LIBS in the configure script:

```
# Define "external" libraries
#---------------------------
# These are the third-party libraries distributed with oomph-lib.
# Note: If you want to link against another, already existing local
# installation of one of these libraries, edit the EXTERNAL_LIBS
# assignment below. As an example, if you want to use your own BLAS
# library, and you'd usually link against this with
#
#    g++ -o my_code my_code.cc -lblas
#
# then replace "-loomph_blas" by "-lblas" below:
#
EXTERNAL_LIBS='-loomph_hsl -loomph_superlu_3.0 -loomph_metis_4.0 -loomph_arpack -loomph_lapack
       -loomph_flapack -loomph_blas'
```

If you wish to link against some other version of the library, edit the EXTERNAL_LIBS variable accordingly.

WARNING: If you work within the autotools framework, remember that the configure script is generated by autoconf, using the file configure.ac, which is itself (re-)generated whenever you run autogen.sh. If you use the autotools, you should edit the EXTERNAL_LIBS variable in the file configure/configure.ac_scripts/start which forms one of the building blocks from which autogen.sh assembles the configure.ac file. Once you have changed the EXTERNAL_LIBS variable in configure/configure.ac_scripts/start you should (re-)run autogen.sh in oomph-lib's top-level directory.

NEW (version 0.85)

You can now use the -with-blas and -with-lapack configure flags to specify the location of your own blas and lapack libraries; see the installation tutorial for details. Equivalent flags will soon be added for the other libraries.

1.1.12 Missing 'this->'

Recent versions of the gcc compilers enforce the C++ standard much more rigorously than earlier versions. Unfortunately, the standard includes some rules that are so counter-intuitive that it is hard get into the habit of using them, especially if code is developed on a compiler that does not enforce the standard as rigorously.

The most frequent problem arises in classes that are derived from a templated base class. The C++ standard insists that all references to member functions (or member data) that is defined in the templated base class must be preceded by "this->" when the reference is made in the derived class. Allegedly, this is necessary to avoid ambiguities, though it is not entirely clear what this ambiguity is supposed to be... Here is a driver code that illustrates the problem.

```
//LIC// ====================================================================
//LIC// This file forms part of oomph-lib, the object-oriented,
```

```
//LIC// multi-physics finite-element library, available
//LIC// at http://www.oomph-lib.org.
//LIC//
//LIC//    Version 1.0; svn revision $LastChangedRevision$
//LIC//
//LIC// $LastChangedDate$
//LIC//
//LIC// Copyright (C) 2006-2016 Matthias Heil and Andrew Hazel
//LIC//
//LIC// This library is free software; you can redistribute it and/or
//LIC// modify it under the terms of the GNU Lesser General Public
//LIC// License as published by the Free Software Foundation; either
//LIC// version 2.1 of the License, or (at your option) any later version.
//LIC//
//LIC// This library is distributed in the hope that it will be useful,
//LIC// but WITHOUT ANY WARRANTY; without even the implied warranty of
//LIC// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
//LIC// Lesser General Public License for more details.
//LIC//
//LIC// You should have received a copy of the GNU Lesser General Public
//LIC// License along with this library; if not, write to the Free Software
//LIC// Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
//LIC// 02110-1301  USA.
//LIC//
//LIC// The authors may be contacted at oomph-lib@maths.man.ac.uk.
//LIC//
//LIC//====================================================================
//Demo code to document problem with missing "this->"
#include<iostream>



//==========Templated_base_class=========================================
/// Some Non-templated base class
//=======================================================================
template<unsigned TEMPLATE_PARAMETER>
class TemplatedBaseClass
{
public:

 /// Empty constructor
 TemplatedBaseClass(){};

 /// Empty virtual constructor
 ~TemplatedBaseClass(){};

 /// Some member function
 void say_hello_world()
  {
   std::cout << "Hello world from base class " << std::endl;
  }

};



//======templated_derived_class==========================================
/// Some templated derived class
//=======================================================================
template<unsigned TEMPLATE_PARAMETER>
class SomeDerivedClass : public virtual TemplatedBaseClass<TEMPLATE_PARAMETER>
{

public:

 // Empty constructor
 SomeDerivedClass(){};

 // Virtual empty constructor
 virtual ~SomeDerivedClass(){};

 /// Some member function
 void output_template_parameter()
  {
   std::cout << "My template parameter is: "
             << TEMPLATE_PARAMETER << std::endl;

   // Now call the function in the base class

#ifdef USE_BROKEN_VERSION

   // This is illegal according to the C++ standard
   say_hello_world();

#else

   // This is stupid but in line with the C++ standard
```

```
   this->say_hello_world();

#endif

  }

};


//======start_of_main=====================================================
/// Driver
//========================================================================
int main()
{

 // Build the templated object:
 SomeDerivedClass<2> object;

 // Get it to output its template parameter and say hello:
 object.output_template_parameter();

} // end of main
```

If you compile this with sufficiently recent versions of the gcc compilers, using the flag -DUSE_BROKEN_VERSION, the compilation will fail with the following error:

```
broken_this_demo.cc: In member function 'void
       SomeDerivedClass<TEMPLATE_PARAMETER>::output_template_parameter()':
broken_this_demo.cc:55: error: there are no arguments to 'say_hello_world' that depend on a template
      parameter, so a declaration of 'say_hello_world' must be available
broken_this_demo.cc:55: error: (if you use '-fpermissive', G++ will accept your code, but allowing the use
      of an undeclared name is deprecated)
```

You may not only stumble across this problem in one of your own codes but it is also possible that some code in the library itself still violates this rule. This is because templated classes are only built when needed and it is conceivable that oomph-lib's suite of self-tests do not instantiate all templated classes that exist in the library. If you encounter any such problems, check if putting a "this->" in front of the function call fixes the problem. If it does, let us know!

1.2 Compilation problems and run-time errors

1.2.1 Warning about "discarded sections" during linking

When linking, some versions of the gcc compiler produce warnings about references to "discarded sections" being referenced. Here's an example:

```
/usr/bin/ld: `.gnu.linkonce.t._ZNK5oomph8QElementILj3ELj3EE14vertex_node_ptERKj' referenced in section
       `.rodata' of /home/mheil/version185/oomph/build/lib/libgeneric.a(Qelements.o): defined in discarded section `.
      gnu.linkonce.t._ZNK5oomph8QElementILj3ELj3EE14vertex_node_ptERKj' of
       /home/mheil/version185/oomph/build/lib/libgeneric.a(Qelements.o)
```

We admit to being slightly baffled by this. Other libraries seem to suffer from the same problem (google for .rodata discarded, say), but as far as we can tell no solution has ever been suggested, nor does one seem to be required. The executable works fine. Upgrade to a newer version of gcc?

1.2.2 My driver code compiles but dies with a segmentation fault

Suggestions:

- Run `Problem::self_test()` before solving the problem. This function performs a large number of sanity checks and reports any inconsistencies in the data structure.

- Recompile the relevant libraries with the PARANOID and RANGE_CHECKING flags set. Segmentation faults are often caused by out-of-bounds access to STL containers. Since $>$ 95% (?) of the containers used in `oomph-lib` are `Vectors` (`oomph-lib's` wrapper to the STL vector class, with optional range checking) they can easily be detected. Make sure to re-compile again with RANGE_CHECKING switched off before you start any production runs – the run-time overheads incurred by range-checking are significant!

- Recompile the relevant libraries and your driver code with the debugging flag ("-g" for the gnu compiler suite) switched on and all optimisation disabled. Re-run the code in a debugger (`gdb` or its GUI-based equivalent `ddd`) to (try to!) find out where the segmentation fault occurred. [**Careful:** If the segmentation fault is caused by a pointer problem, this naive inspection can be quite misleading – tell-tale signs are that the traceback displays a non-sensical call stack, e.g. a function being called "out of nowhere"; variables that have just been given values not existing; etc.]

1.2.3 My driver code runs but it produces incorrect/non-sensical results

Suggestions:

- Have you applied the boundary conditions correctly? Check this by looping over all nodes in your mesh and documenting the pinned-status of their nodal values, using the function `Node::is_pinned(...)`.

- Have you passed the required (function) pointers to the elements? Most element constructors assign default values for any physical parameters, e.g. the Reynolds number in the Navier-Stokes elements. Similarly, most source functions etc. default to zero. For instance `oomph-lib's` Poisson elements solve the Laplace equation unless a function pointer to the source function is specified.

- In time-dependent problems, have you assigned suitable initial conditions? Note that, if you use elements that are based on the ALE formulation (and all time-dependent elements should be!) you must initialise the history values for the nodal positions, even if the mesh is stationary! See the discussion of `oomph-lib's` timestepping procedures in the context of `the unsteady heat equation` for details.

- Have you implemented the relevant "action" functions, such as `Problem::actions_before_implicit_timestep()` to update any time-dependent boundary conditions?

1.2.4 The Newton solver diverges

Suggestions:

- `oomph-lib's` default solver `Problem::newton_solve(...)` will converge quadratically, provided

  – a "good" initial guess for the solution has been assigned,

  or

  – the problem is linear.

  If the Newton solver fails to converge for a nonlinear problem, try to identify a related linear problem and use continuation to generate a sequence of good initial guesses. For instance, to solve the Navier-Stokes equations at a Reynolds number of 500, say, start by solving the problem for zero Reynolds number (in which case the problem becomes linear so that the Newton method converges in one iteration); increase the Reynolds number by 50, say, and re-solve. Repeat this procedure until the desired value of the Reynolds number is reached.

  Note: `oomph-lib` also provides automatic continuation methods, based on Keller's arclength continuation, but at the moment, no tutorials exist for these.

- If you have tried the above and the Newton method fails to converge even for a linear version of your problem, the most likely reasons are that

  1. You have developed a new element and made a mistake in the implementation of the element's Jacobian matrix. To check if this is the case, comment out the function that computes the element's Jacobian matrix, i.e. the element member functions `get_jacobian(...)` or `fill_in_contribution_to_jacobian(...)`. `oomph-lib` will then use the default implementation of these functions in the `GeneralisedElement` base class to compute the Jacobian matrices by finite-differencing. The executable is likely to run more slowly since the finite-difference-based computation is unlikely to be as efficient as the customised implementation for your specific element, but if the Newton method then converges, you know where to look for your bug! You may also want to check for any un-initialised variables. They are the most likely culprits if your code behaves differently at different levels of optimisation as more aggressive optimisation may suppress any default initialisations of data – in fact, you should never rely on that anyway!

  2. Your problem contains "enslaved" variables, such as the nodal positions in a free-boundary problem. If the node update in response to changes in the shape of the domain boundary is performed by an algebraic node update procedure (using `AlgebraicNodes`, `SpineNodes` or nodes whose position is updated by a `MacroElement/Domain` - based procedure), the position of the nodes in the "bulk" mesh must be updated whenever the Newton method updates the unknowns. This is most easily done by calling `Mesh::node_update()` in `Problem::actions_before_newton_convergence_check()`.

1.3 Customisation and optimisation

1.3.1 I don't have tecplot. How do I change oomph-lib's output so it can be displayed by my own plotting package?

oomph-lib's high-level post-processing routines output the results of the computations in a form that is suitable for display with tecplot, a powerful commercial plotting package. Purists may find it odd that an open-source library should choose an output format that is customised for a commercial software package. We tend to agree... Our only excuse is that tecplot is very very good, and without it we would have found it extremely difficult to create many of the plots shown in the tutorials. [If you know of any open-source plotting package whose capabilities are comparable to those of tecplot, let us know!]

Angelo Simone has written a python script that converts oomph-lib's output to the vtu format that can be read by paraview, an open-source 3D plotting package. The conversion script can currently deal with output from meshes that are composed of 2D quad elements – the extension to 3D is work in progress. Use of the conversion script is documented in another tutorial.

It is possible to display oomph-lib's default output (in more elementary form, obviously) with gnuplot. The trick is to specify the using option in gnuplot's plot commands – in this mode gnuplot ignores tecplot's "ZONE" commands. For instance, trying to plot the x-y data created by the demo code for the solution of the 1D Poisson equation with

```
plot "RESLT/soln0.dat"
```

will fail because gnuplot gets confused by the ZONE specifications required by tecplot. However,

```
plot "RESLT/soln0.dat" using 1:2
```

works.

If the data is too complex to be displayed by gnuplot, you may wish to customise the output for your preferred plotting package. This is easily done as oomph-lib creates its output element-by-element. The elements' various output(...) functions are virtual functions that can easily be overloaded in a user-defined wrapper class.

Here is an example driver code that illustrates how to change the output from oomph-lib's QPoisson↩ Element family of 1D-line/2D-quad/3D-brick Poisson elements so that they output the string "Hello world".

We include oomph-lib's generic and poisson library headers:

```
//LIC// ====================================================================
//LIC// This file forms part of oomph-lib, the object-oriented,
//LIC// multi-physics finite-element library, available
//LIC// at http://www.oomph-lib.org.
//LIC//
//LIC//    Version 1.0; svn revision $LastChangedRevision$
//LIC//
//LIC// $LastChangedDate$
//LIC//
//LIC// Copyright (C) 2006-2016 Matthias Heil and Andrew Hazel
//LIC//
//LIC// This library is free software; you can redistribute it and/or
//LIC// modify it under the terms of the GNU Lesser General Public
//LIC// License as published by the Free Software Foundation; either
//LIC// version 2.1 of the License, or (at your option) any later version.
//LIC//
//LIC// This library is distributed in the hope that it will be useful,
//LIC// but WITHOUT ANY WARRANTY; without even the implied warranty of
//LIC// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
//LIC// Lesser General Public License for more details.
//LIC//
//LIC// You should have received a copy of the GNU Lesser General Public
```

```
//LIC// License along with this library; if not, write to the Free Software
//LIC// Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
//LIC// 02110-1301  USA.
//LIC//
//LIC// The authors may be contacted at oomph-lib@maths.man.ac.uk.
//LIC//
//LIC//====================================================================
//Demo code to document customisation of output

// oomph-lib includes
#include "generic.h"
#include "poisson.h"
```

and then create a customised version of the Poisson elements in which we overload the tecplot-based `QPoisson↩ Element<DIM,NNODE_1D>::output(...)` function, defined in the `poisson` library:

```
// The wrapper class for the element has to be included into
// the oomph-lib namespace
namespace oomph
{

//======customised_poisson=============================================
/// Customised Poisson element -- simply overloads the output function.
/// All other functionality is retained.
//=====================================================================
template<unsigned DIM, unsigned NNODE_1D>
class CustomisedQPoissonElement : public virtual QPoissonElement<DIM,NNODE_1D>
{

public:

 /// Empty constructor
 CustomisedQPoissonElement(){};

 /// Empty virtual constructor
 ~CustomisedQPoissonElement(){};

 /// Overload output function
 void output(std::ostream& output_file)
  {
   output_file << "Hello world" << std::endl;
  }

};

} //end extension of oomph-lib namespace
```

If we now call the output function, the version defined in the customised element is used. The remaining implementation of the Poisson element remains unchanged.

```
//======start_of_main==================================================
/// Driver
//=====================================================================
int main()
{

 using namespace oomph;

 // Build the templated object:
 CustomisedQPoissonElement<2,2> element;

 // Call the element's (customised) output function and dump to screen
 element.output(std::cout);

} // end of main
```

1.3.2 oomph-lib's implementation of the Navier-Stokes equations (say) is too general (and therefore too expensive) for my application. How do I change this?

Many of `oomph-lib's` equations classes (or elements) are implemented in great generality. For instance, our discretisation of the Navier-Stokes equations includes a source term in the continuity equation, and body force terms in the momentum equations; it allows switching between the stress-divergence and simplified forms of the viscous terms; it includes the mesh velocity into the ALE formulation of the time-derivatives; etc. This makes the elements very versatile and robust. However, the generality/robustness comes at a price: Even though we provide default values for most functions (e.g. the body force terms default to zero), their evaluation requires a finite amount of CPU time. If you wish to use the elements in a simple application in which the Navier-Stokes equations are solved in a fixed domain, without any body forces or other source terms, say, you may wish to disable the additional functionality.

This is easily done: After all, `oomph-lib` is open-source software and you can therefore change anything you want! In principle, you could edit the source code in the `src/navier_stokes` directory and delete (or at least comment out) all the functionality that you do not require. However, this is probably a risky step as it will break all demo codes (used during `oomph-lib's` self-test procedure) that use some of the features that you are not interested in. We therefore recommend copying the content of the directory `src/navier_stokes` into a new directory, e.g. `user_src/my_navier_stokes` and to edit the copied sources. Follow the instructions on the `oomph-lib` `installation page` to turn these sources into a separate library against which you can link.

1.3.3 How do I build the self tests without running them?

The default behaviour when using `make check` is to compile and run the self tests one test at a time. This means that compilation failures may take a long time to appear which can be frustrating when making changes to the main library. The alternative is to first compile all self tests using

```
make check -k TESTS_ENVIRONMENT=true
```

and then once you are sure everything compiles run the self tests as normal.

Note: this "trick" relies on undocumented behaviour in automake and so it is possible (but unlikely) that it will not work in new versions (tested and working with GNU automake 1.11.3).

1.4 Finding your way around the distribution

1.4.1 There is so much information -- how do I get started?

Yes, oomph-lib does contain a lot of code and a lot of documentation. How to get started obviously depends on your background: Are you familiar with the finite element method? How good is your knowledge of C++? Etc.

Here are some possible "routemaps" around the library:

- **You are familiar with the finite element method and have a fairly good knowledge of C++**
  - Have a look through the list of example codes to get a feeling for oomph-lib's capabilities. Pick a problem that interests you and study the associated tutorial. Copy the driver code into your own directory and play with it.
  - Once you have played with a few example codes, you may wish to to learn more about oomph-lib's overall data structure, or find out how to optimise the library for your particular application.
- **You have never used finite element methods but have a fairly good knowledge of C++**
  - Study the "Top Down" introduction. This document includes a "low tech" overview of the mathematical/theoretical background and contrasts procedural implementations of the finite element method with the object-oriented approach adopted in oomph-lib.
  - Consult the (Not-So-)Quick-Guide to learn how to construct basic oomph-lib objects for your problem: Problems, Meshes, FiniteElements, etc.
  - Continue with the steps suggested above.
- **You have never used finite element methods and are a newcomer to C++**
  - Buy Daoqi Yang brilliant book C++ and Object-Oriented Numeric Computing for Scientists and Engineers. Read it! Pretty much everything in this book is relevant for some parts of oomph-lib. You should at least understand:
    - The procedural aspects of C++ (basic types, functions and control structures).
    - Namespaces.
    - Classes (private, protected and public members; inheritance and multiple inheritance; virtual and pure virtual functions; base classes and derived classes; static and dynamic casts).
    - Templates and template instantiations.

* The standard template library (STL).

– Continue with the steps suggested above.

1.4.2 Where is this class/function/... defined?

Assume you have studied one of the example codes and wish to find out more about the implementation of a particular class or function that is used there. How do you find its source code and/or its full documentation?

Generally, a class/function that is used in a demo code can only be defined in one of two places:

1. In the demo driver code itself.
2. In an included file and/or an associated library.

oomph-lib's tutorials tend to provide a fairly complete annotated listing of the relevant driver codes; if the function you are interested in is not mentioned explicitly in the tutorial, it is most likely to be defined in an include file. You can inspect the driver code in its entirety by following the link at the end of the tutorials. If you cannot find the class/function there, it must be defined in one of the include files listed at the beginning of the source code.

The included files themselves can either be located in the same directory as the demo driver (the directory also tends to be mentioned at the end of the tutorial) or in one of oomph-lib's sub-libraries. The source code for these is located in the sub-directories of the src directory. Often the class/function is defined in a source file with an "obvious" name; if not, use grep to find it. This can, of course, be done recursively. For instance, the command

```
find . \( -name '*.h' -o -name '*.cc' \) -exec grep -H FiniteElement {} \;
```

issued in oomph-lib's top-level directory will search through the entire distribution to locate files that contain the string "FiniteElement".

You can also use the html-based representation of oomph-lib's data structure, created by doxygen, in the "bottom-up" discussion of the data structure. (Note that the search menu may not work on your browser.)

1.5 If all else fails...

1.5.1 How to report problems/bugs

If all else fails and you think you have found a bug in the library, make sure you follow these steps:

1. Isolate the problem: Try to identify the shortest driver code that still produces the problem.

2. Double-check the relevant documentation, the installation instructions and the other FAQs listed here.

3. Does the problem persist when you compile the library and your test code without optimisation, and when the RANGE_CHECKING and PARANOID flags are set?

4. Does the problem occur with a sufficiently recent version of the gcc compiler suite (version 3.2.3 and later)?

5. If the above steps identify the problem, let us know, ideally with a bug fix!

6. If you can't fix the problem yourself, get in touch either directly, or via our bugzilla-based bug-and-feature-tracking system, accessible online at

   http://oomph-lib.maths.man.ac.uk/bugzilla

   and provide as much information as possible (clear description of the problem; the source code; the Makefile; details of the compiler and compilation flags used; any warning/error messages that are displayed during the compilation of the library or the driver code itself; etc.)

1.6 PDF file

A pdf version of this document is available.