DOLFIN User Manual

September 13, 2005

Hoffman, Jansson, Logg

www.fenics.org

Visit http://www.fenics.org/ for the latest version of this manual.
Send comments and suggestions to dolfin-dev@fenics.org.

Contents

About this manual

This manual is currently being written. A first version of this manual should be ready sometime in the fall of 2005.

Intended audience

This manual is written both for the beginning and the advanced user. There is also some useful information for developers. More advanced topics are treated at the end of the manual or in the appendix.

Typographic conventions

- Code is written in monospace (typewriter) `like this`.
- Commands that should be entered in a Unix shell are displayed as follows:

  ```
  # ./configure
  # make
  ```

  Commands are written in the dialect of the `bash` shell. For other shells, such as `tcsh`, appropriate translations may be needed.

Enumeration and list indices

Throughout this manual, elements x_i of sets $\{x_i\}$ of size n are enumarated from $i = 0$ to $i = n - 1$. Derivatives in $\mathbb{R}^n$ are enumerated similarly: $\frac{\partial}{\partial x_0}, \frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_{n-1}}$.

Contact

Comments, corrections and contributions to this manual are most welcome and should be sent to

`dolfin-dev@fenics.org`

Chapter 1

Introduction

FIXME: Automation of CMM, FEniCS, purpose of DOLFIN: PSE for differential equations, C++ interface of FEniCS, etc

1.1 The FEniCS project

FIXME: Automation of CMM, other components of **FEniCS**

1.2 The finite element method

FIXME: Automation of discretization

1.3 Overview

FIXME: Component diagram, user, module, kernel

Chapter 2

Quickstart

2.1 Downloading DOLFIN

2.2 Installing DOLFIN

2.3 Solving Poisson's equation with DOLFIN

Chapter 3

Linear algebra

FIXME: Write about the wrappers, PETSc, using mat() and vec() to do more advanced operations with PETSc etc.

Chapter 4

Functions

FIXME: Discuss the Function class and the different representations.

Chapter 5

The mesh

FIXME: Triangular, tetrahedral, include some images, mesh refinement, connectivity, iterators, file formats, local ordering

Chapter 6

Ordinary differential equations

FIXME: Mono-adaptive, multi-adaptive, ODE base class, simple example, error control, adaptivity, complex ODE, implicit, homotopies

Chapter 7

Partial differential equations

FIXME: Variational formulation, examplify with Poisson, FFC, finite elements, FIAT, assembly, functionals

Chapter 8

Input/output

8.1 Pre- and post-processing

FIXME: DOLFIN relies on external programs for pre- and post-processing

8.2 Files and objects

FIXME: Discuss operators >> and <<

8.3 File formats

FIXME: Insert table here of filename suffixes and corresponding formats.

8.3.1 DOLFIN XML

FIXME: The native format

8.3.2 Another format

8.3.3 Another format

8.3.4 Another format

8.4 Adding a new file format

FIXME: Discuss classes File, GenericFile etc

Chapter 9

The log system

FIXME: This needs to be rewritten

The purpose of the log system is to provide a simple and clean interface for logging messages, including warnings and errors.

The following functions / macros are provided for logging:

```
dolfin_info();
dolfin_debug();
dolfin_warning();
dolfin_error();
dolfin_assert();
```

Examples of usage:

```
dolfin_info("Created vector of size %d.", x.size());
dolfin_debug("Opened file");
dolfin_warning("Unknown cell type.");
dolfin_error("Out of memory.");
dolfin_assert(i < m);
```

Note that in order to pass additional arguments to the last three functions (which are really macros, in order to automatically print information about file names, line numbers and function names), the variations `dolfin_debug1()`, `dolfin_debug2()` and so on, must be used.

As an alternative to `dolfin_info()`, C++ style output to cout (`dolfin::cout`, and not `std::cout`) can be used. These messages will be delivered to the same destination as messages by use of the function `dolfin_info()`.

Examples of usage:

```
cout << "Assembling matrix: " << A << endl;
cout << "Refining grid: " << grid << endl;
```

The `dolfin_assert()` macro should be used for simple tests that may occur often, such as checking indexing in vectors. The check is turned on only if DEBUG is defined.

To notify progress by a progress session, use the class Progress.

Examples of usage:

```
Progress p("Assembling", grid.noCells());

for (CellIterator c(grid); !c.end(); ++c)     ...    p++;
```

Progress also supports the following usage:

```
p = i;    // Specify step number
p = 0.5;  // Specify percentage
p.update(t/T, "Time is t = %f", t);
```

Chapter 10

Parameters

FIXME: Parameter database, setting values, getting values, defining new parameters

Chapter 11

Solvers

FIXME: List solvers, then present in detail, include lots of nice images with solver output

11.1 Poisson's equation

Write introduction here, equations etc.

11.1.1 Usage

Present API of solver and give an example.

11.1.2 Performance

Write something about the performance of the solver.

11.1.3 Limitations

Write something about the limitations of the solver.

11.2 Convection–diffusion

Write introduction here, equations etc.

11.2.1 Usage

Present API of solver and give an example.

11.2.2 Performance

Write something about the performance of the solver.

11.2.3 Limitations

Write something about the limitations of the solver.

11.3 Incompressible Navier–Stokes

Write introduction here, equations etc.

11.3.1 Usage

Present API of solver and give an example.

11.3.2 Performance

Write something about the performance of the solver.

11.3.3 Limitations

Write something about the limitations of the solver.

11.4 Elasticity

Write introduction here, equations etc.

11.4.1 Usage

Present API of solver and give an example.

11.4.2 Performance

Write something about the performance of the solver.

11.4.3 Limitations

Write something about the limitations of the solver.

Appendix A

Reference elements

A.1 The reference triangle

The reference triangle (Figure A.1) is defined by the following three vertices:

$$\begin{aligned} v^0 &= (0, 0), \\ v^1 &= (1, 0), \\ v^2 &= (0, 1). \end{aligned} \tag{A.1}$$

Note that this corresponds to a counter-clockwise orientation of the vertices in the plane.

The edges of the reference triangle are ordered following the convention that edge e^i should be opposite to vertex v^i for $i = 0, 1, 2$, with the vertices of each edge ordered to give a counter-clockwise orientation of the triangle in the plane:

$$\begin{aligned} e^0 &: (v^1, v^2), \\ e^1 &: (v^2, v^0), \\ e^2 &: (v^0, v^1). \end{aligned} \tag{A.2}$$

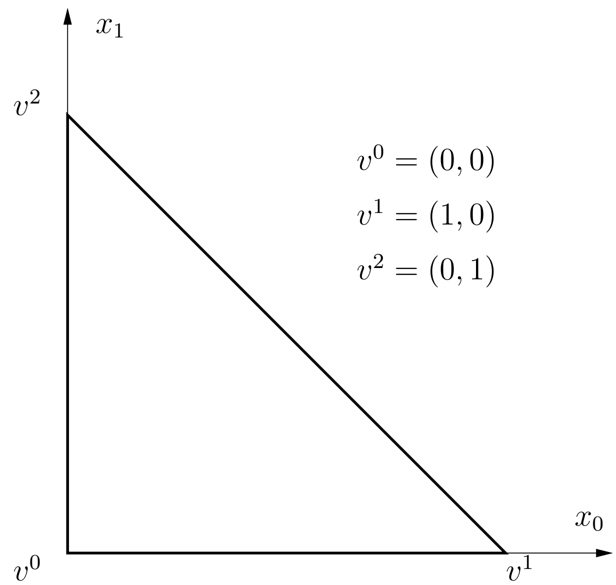

Figure A.1: Physical coordinates of the reference triangle.

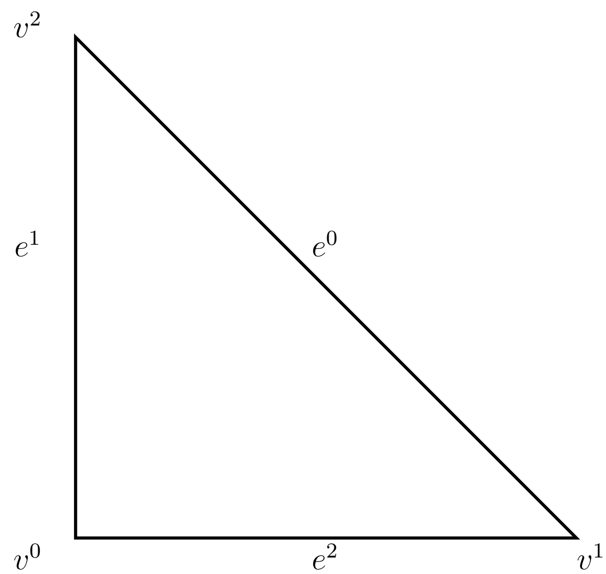

Figure A.2: Ordering of mesh entities (vertices and edges) for the reference triangle.

A.2 The reference tetrahedron

The reference tetrahedron (Figure A.3) is defined by the following four vertices:

$$
\begin{aligned}
v^0 &= (0,0,0), \\
v^1 &= (1,0,0), \\
v^2 &= (0,1,0), \\
v^4 &= (0,0,1).
\end{aligned}
\tag{A.3}
$$

The faces of the reference tetrahedron are ordered following the convention that face f^i should be opposite to vertex v^i for $i = 0,1,2,3$, with the vertices of each face ordered to give a counter-clockwise orientation of each face as seen from the outside of the tetrahedron and the first vertex of face f^i given by vertex $v^{i+1 \mod 4}$:

$$
\begin{aligned}
f^0 &: (v^1, v^3, v^2), \\
f^1 &: (v^2, v^3, v^0), \\
f^2 &: (v^3, v^1, v^0), \\
f^3 &: (v^0, v^1, v^2).
\end{aligned}
\tag{A.4}
$$

The edges of the reference tetrahedron are ordered following the convention that edges e^0, e^1, e^2 should correspond to the edges of the reference triangle. Edges e^3, e^4, e^5 all ending up at vertex v^3 are ordered based on their first vertex:

$$
\begin{aligned}
e^0 &: (v^1, v^2), \\
e^1 &: (v^2, v^0), \\
e^2 &: (v^0, v^1), \\
e^3 &: (v^0, v^3), \\
e^4 &: (v^1, v^3), \\
e^5 &: (v^2, v^3).
\end{aligned}
\tag{A.5}
$$

The ordering of vertices on faces implicitly defines an ordering of edges on faces by identifying an edge on a face with the opposite vertex on the face:

$$\begin{aligned} f^0 &: (e^5, e^0, e^4), \\ f^1 &: (e^3, e^1, e^5), \\ f^2 &: (e^2, e^3, e^4), \\ f^3 &: (e^0, e^1, e^2). \end{aligned} \tag{A.6}$$

Note that the ordering of edges on f^3 is the same as the ordering of edges on the reference triangle. Also note that the internal ordering of vertices on edges does not always follow the orientation of the face (which is not possible).

A.3 Ordering of degrees of freedom

The local and global orderings of degrees of freedom or *nodes* are obtained by associating each node with a mesh entity, locally and globally.

A.3.1 Mesh entities

We distinguish between mesh entities of different topological dimensions:

vertices	topological dimension 0
edges	topological dimension 1
faces	topological dimension 2
cells	topological dimension 2 or 3

A cell can be either a triangle or a tetrahedron depending on the type of mesh. For a mesh consisting of triangles, the mesh entities involved are vertices, edges and cells, and for a mesh consisting of tetrahedrons, the mesh entities involved are vertices, edges, faces and cells.

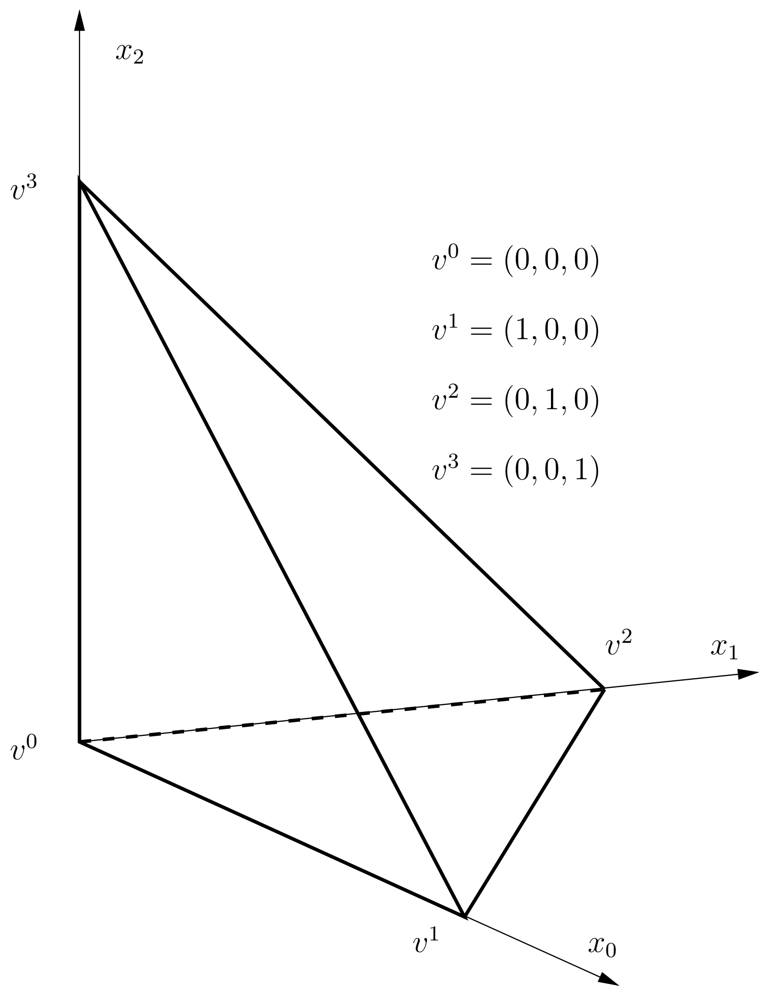

Figure A.3: Physical coordinates of the reference tetrahedron.

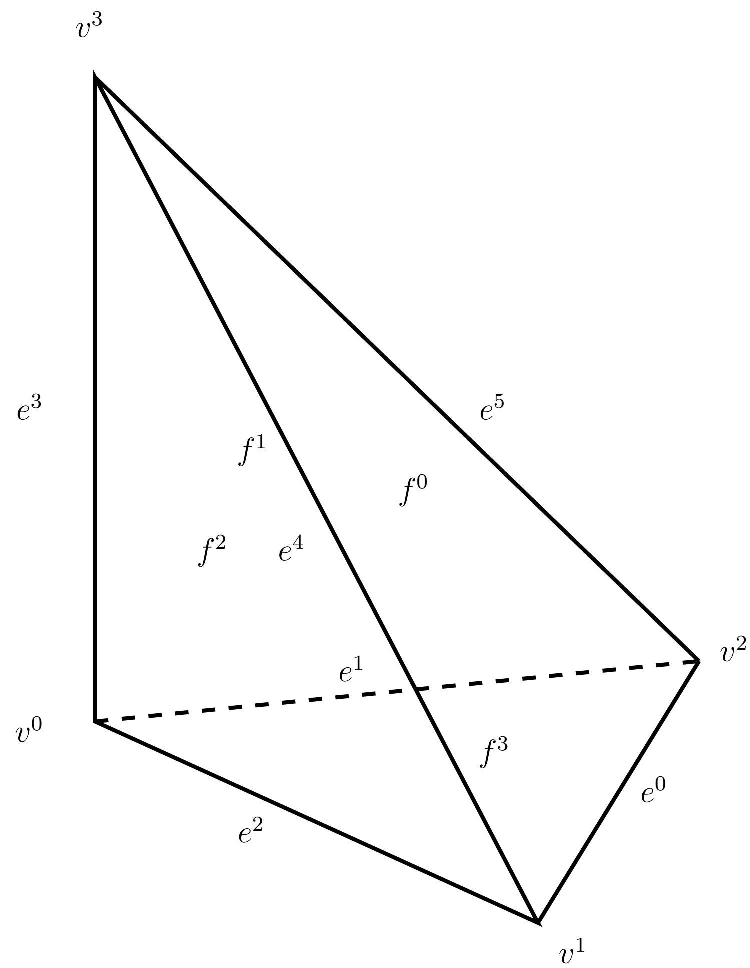

Figure A.4: Ordering of mesh entities (vertices, edges, faces) for the reference tetrahedron.

A.3.2 Ordering among mesh entities

With each mesh entity, there can be associated zero or more nodes and the nodes are ordered locally and globally based on the topological dimension of the mesh entity with which they are associated. Thus, any nodes associated with vertices are ordered first and nodes associated with cells last.

If more than one node is associated with a single mesh entity, the internal ordering of the nodes associated with the mesh entity becomes important, in particular for edges and faces, where the nodes of two adjacent cells sharing a common edge or face must lign up.

A.3.3 Internal ordering on edges

For edges containing more than one node, the nodes are ordered in the direction from the first vertex (v_e^0) of the edge to the second vertex (v_e^1) of the edge as in Figure A.5.

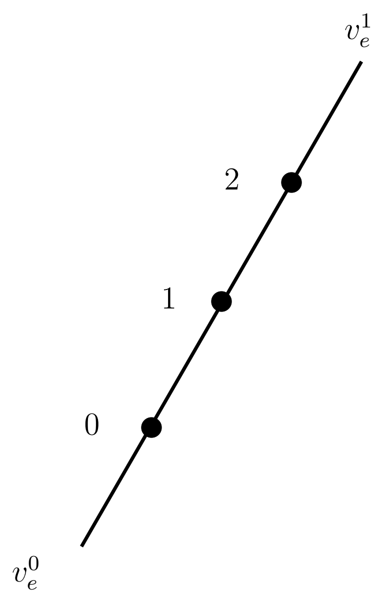

Figure A.5: Internal ordering of nodes on edges.

A.3.4 Alignment of edges

Depending on the orientation of any given cell, an edge on the cell may be aligned or not aligned with the corresponding edge on the reference cell if the vertices of the cell are mapped to the reference cell. We define the *alignment* of an edge with respect to a cell to be 0 if the edge is aligned with the orientation of the reference cell and 1 otherwise.

Example 1: The alignment of the first edge (e^0) on a triangle is 0 if the first vertex of the edge is the second vertex (v^1) of the triangle.

Example 2: The alignment of the second edge (e^1) on a tetrahedron is 0 if the first vertex of the edge is the third vertex (v^2) of the tetrahedron.

If two cells share a common edge and the edge is aligned with one of the cells and not the other, we must reverse the order in which the local nodes are mapped to global nodes on one of the two cells. As a convention, the order is kept if the alignment is 0 and reversed if the alignment is 1.

A.3.5 Internal ordering on faces

For faces containing more than one node, the ordering of nodes is nested going from the first to the third vertex and in each step going from the first to the second vertex as in Figure A.6.

A.3.6 Alignment of faces

There are six different ways for a face to be aligned on a tetrahedron; there are three ways to pick the first edge of the face, and once the first edge is picked, there are two ways to pick the second edge. To define an alignment of faces as an integer between 0 and 5, we compare the ordering of edges on a face with the ordering of edges on the corresponding face on the reference tetrahedron. If the first edge of the face matches the first edge on the corresponding face on the reference tetrahedron and also the second edge matches the second edge on the reference tetrahedron, then the alignment is 0. If only the first

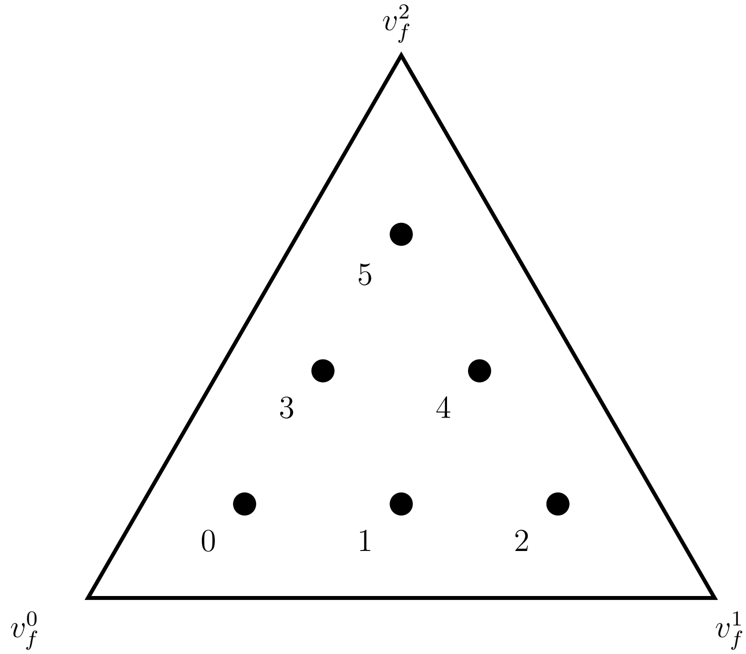

Figure A.6: Internal ordering of nodes on faces.

edge matches, then the alignment is 1. We similarly define alignments $2, 3$ by matching the first and second edges with the second and third edges on the corresponding face on the reference tetrahedron, and alignments $4, 5$ by matching the first and second edges with the third and first edges on the corresponding face on the reference tetrahedron.

Example 1: The alignment of the first face of a tetrahedron is 0 if the first edge of the face is edge number 5 and the second edge is edge number 0.

Example 2: The alignment of the first face of a tetrahedron is 1 if the first edge of the face is edge number 5 and the second edge is not edge number 0. (It must then be edge number 4.)

Example 3: The alignment of the first face of a tetrahedron is 4 if the first edge of the face is edge number 4 and the second edge is edge number 5.

Example 4: The alignment of the first face of a tetrahedron is 5 if the first edge of the face is edge number 4 and the second edge is not edge number 5. (It must then be edge number 0.)

Appendix B

Installation

The source code of **DOLFIN** is portable and should compile on any Unix system, although it is developed mainly under GNU/Linux (in particular Debian GNU/Linux). Questions, bug reports and patches concerning the installation should be directed to the **DOLFIN** mailing list at the address

```
dolfin-dev@fenics.org
```

DOLFIN must currently be compiled directly from source, but effort is underway to provide precompiled Debian packages of **DOLFIN** and other **FEniCS** components.

B.1 Installing from source

B.1.1 Dependencies and requirements

DOLFIN depends on a number of libraries that need to be installed on your system. These libraries include Libxml2 and PETSc. In addition to these libraries, you need to install **FIAT** and **FFC** if you want to define your own variational forms.

Installing Libxml2

Libxml2 is a library used by **DOLFIN** to parse XML data files. Libxml2 can be obtained from

```
http://xmlsoft.org/
```

For Debian users, the package to install is `libxml2-dev`.

Installing PETSc

PETSc is a library for the solution of linear and nonlinear systems, functioning as the backend for the **DOLFIN** linear algebra classes. **DOLFIN** depends on PETSc version 2.3.0, which can be obtained from

```
http://www-unix.mcs.anl.gov/petsc/petsc-2/
```

Follow the installation instructions on the PETSc web page. Normally, you should only have to perform the following simple steps in the PETSc source directory:

```
# export PETSC_DIR=‘pwd‘
# ./config/configure.py --with-clanguage=cxx --with-shared=1
# make all
```

Add `--download-hypre=yes` to configure.py if you want to install Hypre which provides a collection of preconditioners, including algebraic multigrid (AMG).

DOLFIN assumes that `PETSC_DIR` is `/usr/local/lib/petsc/` but this can be controlled using the flag `--with-petsc-dir=<path>` when configuring DOLFIN (see below).

Installing FFC

DOLFIN uses the FEniCS Form Compiler **FFC** to process variational forms. **FFC** can be obtained from

```
http://www.fenics.org/
```

Follow the installation instructions given in the **FFC** manual. **FFC** follows the standard for Python packages, which means that normally you should only have to perform the following simple step in the **FFC** source directory:

```
# python setup.py install
```

Note that **FFC** depends on **FIAT** , which in turn depends on the Python packages Numeric (Debian package `python-numeric`) and LinearAlgebra (Debian package `python-numeric-ext`). Refer to the **FFC** manual for further details.

B.1.2 Downloading the source code

The latest release of **DOLFIN** can be obtained as a `tar.gz` archive in the download section at

```
http://www.fenics.org/
```

Download the latest release of **DOLFIN**, for example `dolfin-0.1.0.tar.gz`, and unpack using the command

```
# tar zxfv dolfin-0.1.0.tar.gz
```

This creates a directory `dolfin-0.1.0` containing the **DOLFIN** source code.

If you want the very latest version of **DOLFIN**, there is also a version named `dolfin-cvs-current.tar.gz` which provides a snapshot of the current CVS version of **DOLFIN**, updated automatically from the CVS repository each hour. This version may contain features not yet present in the latest release, but may also be less stable and even not work at all.

B.1.3 Compiling the source code

DOLFIN is built using the standard GNU Autotools (Automake, Autoconf), which means that the installation procedure is simple:

```
# ./configure
# make
```

followed by an optional

```
# make install
```

to install **DOLFIN** on your system.

The configure script will check for a number of libraries and try to figure out how compile **DOLFIN** against these libraries. The configure script accepts a collection of optional arguments that can be used to control the compilation process. A few of these are listed below. Use the command

```
# ./configure --help
```

for a complete list of arguments.

- Use the option `--prefix=<path>` to specify an alternative directory for installation of **DOLFIN**. The default directory is `/usr/local/`, which means that header files will be installed under `/usr/local/inlude/` and libraries will be installed under `/usr/local/lib/`. This option can be useful if you don't have root access but want to install **DOLFIN** locally on a user account as follows:

```
# mkdir ~/local
# ./configure --prefix=~/local
# make
# make install
```

- Use the option `--enable-debug` to compile **DOLFIN** with debugging symbols and assertions.
- Use the option `--enable-optimization` to compile an optimized version of **DOLFIN** without debugging symbols and assertions.
- Use the option `--disable-curses` to compile **DOLFIN** without the curses interface (a text-mode graphical user interface).
- Use the option `--disable-mpi` to compile **DOLFIN** without support for MPI (Message Passing Interface), assuming PETSc has been compiled without support for MPI.
- Use the option `--with-petsc-dir=<path>` to specify the location of the PETSc directory. By default, **DOLFIN** assumes that PETSc has been installed in `/usr/local/lib/petsc/`.

B.1.4 Compiling the demo programs

After compiling the **DOLFIN** library according to the instructions above, you may want to try one of the demo programs in the subdirectory `src/demo/` of the **DOLFIN** source tree. Just enter the directory containing the demo program you want to compile and type `make`. You may also compile all demo programs at once using the command

```
# make demo
```

B.1.5 Compiling a program against DOLFIN

Whether you are writing your own Makefiles or using an automated build system such as GNU Autotools or BuildSystem, it is straightforward to compile a program against **DOLFIN**. The necessary include and library paths can be obtained through the script `dolfin-config` which is automatically generated during the compilation of **DOLFIN** and installed in the `bin` subdirectory of the `<path>` specified with `--prefix`. Assuming this directory is in your executable path (environment variable `PATH`), the include path for building **DOLFIN** can be obtained from the command

```
dolfin-config --cflags
```

and the path to **DOLFIN** libraries can be obtained from the command

```
dolfin-config --libs
```

If `dolfin-config` is not in your executable path, you need to provide the full path to `dolfin-config`.

Examples of how to write a proper `Makefile` are provided with each of the example programs in the subdirectory `src/demo/` in the **DOLFIN** source tree.

B.2 Debian package

In preparation.

Appendix C

Contributing code

If you have created a new module, fixed a bug somewhere, or have made a small change which you want to contribute to **DOLFIN**, then the best way to do so is to send us your contribution in the form of a patch. A patch is a file which describes how to transform a file or directory structure into another. The patch is built by comparing a version which both parties have against the modified version which only you have.

C.1 Creating a patch

The tool used to create a patch is called `diff` and the tool used to apply the patch is called `patch`. These tools are free software and are standard on most Unix systems.

Here's an example of how it works. Start from the latest release of **DOLFIN**, which we here assume is release 0.1.0. You then have a directory structure under `dolfin-0.1.0` where you have made modifications to some files which you think could be useful to other users.

1. Clean up your modified directory structure to remove temporary and binary files which will be rebuilt anyway:

```
# make clean
```

2. From the parent directory, rename the **DOLFIN** directory to something else:

```
# mv dolfin-0.1.0 dolfin-0.1.0-mod
```

3. Unpack the version of **DOLFIN** that you started from:

```
# tar zxfv dolfin-0.1.0.tar.gz
```

4. You should now have two **DOLFIN** directory structures in your current directory:

```
# ls
dolfin-0.1.0
dolfin-0.1.0-mod
```

5. Now use the `diff` tool to create the patch:

```
# diff -u --new-file --recursive dolfin-0.1.0
  dolfin-0.1.0-mod > dolfin-<identifier>-<date>.patch
```

written as one line, where `<identifier>` is a keyword that can be used to identify the patch as coming from you (your username, last name, first name, a nickname etc) and `<date>` is today's date in the format `yyyy-mm-dd`.

6. The patch now exists as `dolfin-<identifier>-<date>.patch` and can be distributed to other people who already have `dolfin-0.1.0` to easily create your modified version. If the patch is large, compressing it with for example `gzip` is advisable:

```
# gzip dolfin-<identifier>-<date>.patch
```

C.2 Sending patches

Patch files should be sent to the **DOLFIN** mailing list at the address dolfin-dev@fenics.org

Include a short description of what your patch accomplishes. Small patches have a better chance of being accepted, so if you are making a major contribution, please consider breaking your changes up into several small self-contained patches if possible.

C.3 Applying a patch (maintainers)

Let's say that a patch has been built relative to **DOLFIN** release 0.1.0. The following description then shows how to apply the patch to a clean version of release 0.1.0.

1. Unpack the version of **DOLFIN** which the patch is built relative to:

```
# tar zxfv dolfin-0.1.0.tar.gz
```

2. Check that you have the patch dolfin-<identifier>-<date>.patch and the **DOLFIN** directory structure in the current directory:

```
# ls
dolfin-0.1.0
dolfin-<identifier>-<date>.patch
```

   Unpack the patch file using gunzip if necessary.

3. Enter the **DOLFIN** directory structure:

```
# cd dolfin-0.1.0
```

4. Apply the patch:

```
# patch -p1 < ../dolfin-<identifier>-<date>.patch
```

   The option -p1 strips the leading directory from the filename references in the patch, to match the fact that we are applying the patch from inside the directory. Another useful option to patch is --dry-run which can be used to test the patch without actually applying it.

5. The modified version now exists as dolfin-0.1.0.

C.4 License agreement

By contributing a patch to **DOLFIN**, you agree to license your contributed code under the GNU General Public License (a condition also built into the GPL license of the code you have modified). Before creating the patch, please update the author and date information of the file(s) you have modified according to the following example:

```
// Copyright (C) 2004-2005 Johan Hoffman and Anders Logg.
// Licensed under the GNU GPL Version 2.
//
// Modified by Johan Jansson 2005.
// Modified by Garth N. Wells 2005.
//
// First added:  2004-06-22
// Last changed: 2005-09-01
```

As a rule of thumb, the original author of a file holds the copyright.

Appendix D

License

DOLFIN is licensed under the GNU General Public License (GPL) version 2, included verbatim below.

```
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
```

```
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)
```

```
These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
```

control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you

may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Index