FEEL MANUAL
A LIBRARY FOR
FINITE AND SPECTRAL ELEMENT METHODS IN
1D, 2D AND 3D

Version 0.91.0 svn7060

Christophe PRUD'HOMME
christophe.prudhomme@ujf-grenoble.fr

Contents

Chapter 1

Building and Programming Environment

1.1 Building Feel

1.1.1 Getting the source via an archive

Feel is distributed as a tarball once in a while. The tarballs are available at

http://ljkforge.imag.fr/feel

Download the latest tarball. Then follow the steps and replace `x`,`y`,`z` with the corresponding numbers

```
tar xzf feel-x.y.z.tar.gz
cd feel-x.y.z
```

1.1.2 Getting the source via Subversion

Creating RSA keys In order to download the sources of Feel, you have to go in LJKForge website (https://ljkforge.imag.fr) and create an account. After the administrator approval, you have to demand the rights to see the project tree.

Then, you will have to create RSA keys to be able to connect to the server using ssh. To do that you have to type the commands : `ssh-keygen -t dsa` and `ssh-keygen -t rsa` to create the keys. After that, you have to copy the `id_dsa.pub` and `id_rsa.pub` files in the My Page > Account Maintenance > Edit SSH Keys section of the LJKForge website. Those files are located in the `~/.ssh/` folder of your computer. You will be able to connect to the server within an hour.

Important : If you don't have the same login on your computer as on LJKForge, you must add the commands

```
host ljkforge.imag.fr
  user <your_login_ljkforge>
```

in the `~/.ssh/config` file.

Downloading the sources

To be able to download the Feel sources, you need subversion and SSH > 1.xxx installed on your computer. In a command prompt, go where you want Feel to be downloaded and type the following command. `svn co svn+ssh://login@ljkforge.imag.fr/svn/feel/feel/trunk feel` where `login` is your login name in the LJKForge plateform. Then, if you want to download the feel-test sources type :

```
cd feel/benchmarks
svn co svn+ssh://login@ljkforge.imag.fr/svn/feel-test/feel-test/trunk validation
```

1.1.3 Dealing with software dependencies

In order to install Feel, you have to install many dependencies before. Those libraries and programs are necessary for the compilation and installation of the Feel librairies.

This is the list of all the librairies you must have installed on your computer, and the `*-dev` packages for some of them.

Here is the list of required packages:

- g++ (>=4.4)
- MPI : openmpi (preferred) or mpich
- Boost (>=1.39)
- Petsc (>=2.3.3)
- Cmake (>=2.6)
- Gmsh[1]
- Libxml2

Here is the list of optional packages:

- Eigen2
- Superlu
- Suitesparse(umfpack)
- Metis: scoth with the metis interface (preferred), metis (non-free)
- Trilinos (>=8.0.8)
- Google perftools
- Paraview[2], this is not stricly required to run Feel programs but it is somehow necessary for visualisation
- Python (>= 2.5) for the validation tools

Note that all these packages are available under Debian/GNU/Linux and Ubuntu. They should be available

1.1.4 Compiling Feel with the CMake

Feel build system supports `cmake` [3]. This should become the preferred way to build Feel as it is much simpler and more powerful in many ways than the autotools.

Feel, using `cmake`, can be built either in source and out of source and different build type:

- minsizerel : minimal size release
- release release
- debug : debug
- none(default)

[1] Gmsh is a pre/post processing software for scientific computing available at `http://www.geuz.org/gmsh`

[2] Paraview is a few parallel scientific data visualisation plateform, `http://www.paraview.org`

[3] `http://www.cmake.org`

CMake In Source Build This is not advised, yoiu should consider out source builds, see next paragraph. Enter the source tree and type

```
cmake .
make
```

To customize or change some build setting one can use the `cmake` curse interface `ccmake`

```
ccmake . # configure and generate
make
```

CMake Out Source Build Create a build directory

```
mkdir feel.opt
cd feel.opt
cmake <directory where the feel source are>
# e.g cmake ../feel if feel.opt is at the same
# directory level as feel
make
```

you can customize the build type:

```
# Debug build type (-g...)
cmake -D CMAKE_BUILD_TYPE=Debug
# Release build type (-O3...)
cmake -D CMAKE_BUILD_TYPE=Release
...
```

Compiling the Feel tutorial

If the command `make check` has been run before the tutorial should be already compiled and ready. The steps are as follows to build the Feel tutorial

```
cd opt/doc/tutorial
make check
```

Here is what the directory should look like

```
cd opt/doc/tutorial
ls

laplacian      Makefile       myintegrals   mymesh        pngs/
tutorial.blg   tutorial.out   tutorial.toc  laplacian.o   myapp
myintegrals.o  mymesh.o       stokes.assert tutorial.aux  pdfs/ styles/
stokes         stokes.o       tutorial.bbl  tutorial.log  tutorial.pdf
```

1.2 Programming environment

1.3 Namepaces

- `Feel`
- `Feel::po`
- `Feel::mpl`
- `Feel::ublas`

- `Feel::math`
- `Feel::fem`
- `Feel::vf`

1.4 Libraries

- `feel/feelcore`
- `feel/feelalg`
- `feel/feelpoly`
- `feel/feeldiscr`
- `feel/feelfilters`
- `feel/feelvf`

Chapter 2

Tutorial

2.1 Creating applications

myapp.cpp

2.1.1 Application and Options

As a Feel user, the first step in order to use Feel is to create an application. First we include the `Application` header file, `feel/feelcore/application.hpp` and the header which the internal Feel options. Feel uses the `boost::program_options`[1] library from Boost to handle its command line options

```
#include <feel/options.hpp>
#include <feel/feelcore/feel.hpp>
#include <feel/feelcore/application.hpp>
```

Next to ease the programming and reading, we use the **using** C++ directive to bring the namespace Feel to the current namespace

```
  using namespace Feel;
```

Then we define the command line options that the applications will provide. Note that on the **return** line, we incorporate the options defined internally in Feel.

```
inline
po::options_description
makeOptions()
{
    po::options_description myappoptions("MyApp options");
    myappoptions.add_options()
        ("dt", po::value<double>()->default_value( 1 ), "time step value")
        ;

    // return the options myappoptions and the feel_options defined
    // internally by Feel
    return myappoptions.add( feel_options() );
}
```

In the example, we provide the options `dt` which takes an argument, a **double** and its default value is `1` if the options is not set by the command line.

Then we describe the application by defining a class `AboutData` which will be typically used by the `help` command line options to describe the application

```
inline
AboutData
makeAbout()
{
    AboutData about( "myapp" ,
                     "myapp" ,
                     "0.1",
                     "my first Feel application",
                     AboutData::License_GPL,
```

[1] http://www.boost.org/doc/html/program_options.html

```
                      "Copyright (c) 2008 Universite Joseph Fourier");

    about.addAuthor("Christophe Prud'homme",
                    "developer",
                    "christophe.prudhomme@ujf-grenoble.fr", "");
    return about;
}
```

Now we turn to the class `MyApp` itself: it derives from `Feel::Application`. Two constructors take `argc`, `argv` and the `AboutData` as well as possibly the description of the command line options `Feel::po::option_description`.

The class `MyApp` must redefine the `run()` member function. It is defined as a pure virtual function in `Application`.

```
class MyApp: public Application
{
public:

    /**
     * constructor only about data and no options description
     */
    MyApp( int argc, char** argv, AboutData const& );

    /**
     * constructor about data and options description
     */
    MyApp( int argc, char** argv,
           AboutData const&,
           po::options_description const&  );

    /**
     * This function is responsible for the actual work done by MyApp.
     */
    void run();
};
```

The implementation of the constructors is usually simple, we pass the arguments to the super class `Application` that will analyze them and subsequently provide them with a `Feel::po::variable_map` data structure which operates like a `map`. Have a look at the document `boost::program_options` for further details.

```
MyApp::MyApp(int argc, char** argv,
             AboutData const& ad )
    :
    Application( argc, argv, ad )
{}
MyApp::MyApp(int argc, char** argv,
             AboutData const& ad,
             po::options_description const& od )
    :
    Application( argc, argv, ad, od )
{}
```

The `run()` member function holds the application commands/statements. Here we provide the smallest code unit: we print the description of the application if the `--help` command line options is set.

```
void MyApp::run()
{
    /**
     * print the help if --help is passed as an argument
     */
    /** \code */
    if ( this->vm().count( "help" ) )
        {
            std::cout << this->optionsDescription() << "\n";
            return;
        }
    /** \endcode */

    /**
```

```
     * store all subsequent data files in a HOME/feel/doc/tutorial/myapp/
     */
    /** \code */
    //# marker8 #
    this->changeRepository( boost::format( "doc/tutorial/%1%/" )
                            % this->about().appName() );
    //# endmarker8 #
    /** \endcode */

    /**
     * print some information that will be written in the log file in
     * HOME/feel/doc/tutorial/myapp/myapp-1.0
     */
    /** \code */
    Log() << "the value of dt is " << this->vm()["dt"].as<double>() << "\n";
    /** \endcode */
}
```

Finally the `main()` function can be implemented. We pass the results of the `makeAbout()` and `makeOptions()` to the constructor of `MyApp` as well as `argc` and `argv`. Then we call the `run()` member function to execute the application.

```
int main( int argc, char** argv )
{
    Feel::Environment env( argc, argv );

    /**
     * intantiate a MyApp class
     */
    /** \code */
    MyApp app( argc, argv, makeAbout(), makeOptions() );
    /** \endcode */

    /**
     * run the application
     */
    /** \code */
    app.run();
    /** \endcode */
}
```

After compiling `myapp`, we can execute it

```
> myapp --help
myapp: my first Feel application
Allowed options:

MyApp options:
  --dt arg (=1)                              time step value

>./myapp --authors
myapp: my first Feel application
          Author Name       Task                          Email Address
-------------------------------------------------------------------------
Christophe Prud'homme   developer  christophe.prudhomme@ujf-grenoble.fr
```

2.1.2 Application, Logging, Archiving, Configuring

Feel provides some basic logging and archiving support: using the `changeRepository` member functions of the class `Application`, the logfile and results of the application will be stored in a subdirectory of `~/feel`. For example the following code

```
    this->changeRepository( boost::format( "doc/tutorial/%1%/" )
                            % this->about().appName() );
```

will create the directory `~/feel/myapp` and will store the logfile and any files created after calling `changeRepository`. Refer to the documentation of `Boost::format` of further details about the arguments to be passed to `changeRepository`. The logfile is named `~/feel/myapp/myapp-1.0`. The name

of the logfile is built using the application name, here `myapp`, the number of processes, here 1 and the id of the current process, here 0.

```
> myapp
> cat ~/feel/myapp/myapp-1.0
myapp-1.0 is opened for debug
[Area 0] the value of dt is 1

> myapp --dt=0.1
> cat ~/feel/myapp/myapp-1.0
myapp-1.0 is opened for debug
[Area 0] the value of dt is 0.1
```

MPI Application

`myapp.cpp`

Feel relies on MPI for parallel computations and the class `Application` initialises the MPI environment.

```
> mpirun -np 2 mympiapp
> cat ~/feel/mympiapp/mympiapp-2.0
mympiapp-2.0 is opened for debug
[Area 0] the value of dt is 1
[Area 0] we are on processor eta
[Area 0] this is process number 0 out of 2
> cat ~/feel/mympiapp/mympiapp-2.1
mympiapp-2.1 is opened for debug
[Area 0] the value of dt is 1
[Area 0] we are on processor eta
[Area 0] this is process number 1 out of 2

> mpirun -np 2 mympiapp --dt=0.01
> cat ~/feel/mympiapp/mympiapp-2.0
mympiapp-2.0 is opened for debug
[Area 0] the value of dt is 0.01
[Area 0] we are on processor eta
[Area 0] this is process number 0 out of 2
> cat ~/feel/mympiapp/mympiapp-2.1
mympiapp-2.1 is opened for debug
[Area 0] the value of dt is 0.01
[Area 0] we are on processor eta
[Area 0] this is process number 1 out of 2
```

Configuration files

Each application can be configured via the command ling but also using a `.cfg` file. If they exist they may have been installed on your system along with Feel or you may create your own configuration files. Feel provides a way to look for them and parse them.

The `.cfg` file is searched in the following order

1. look in the current directory
2. look in the directory `$HOME/feel/config/`
3. look in the directory `$INSTALL_PREFIX/share/feel/config/`, *e.g.* in Debian `/usr/share/feel/config/`

The name of the file can be constructed in two ways `<appname>.cfg` and `feel_<appname>.cfg` where `<appname>` is the string given in the `AboutData` data structure passed to the construction of the `Application` class.

2.1.3 Initializing PETSc and Trilinos

Feel supports also the PETSc and Trilinos framework, the class `Application` takes care of initialize the associated environments.

2.2 Mesh Manipulation

mymesh.cpp

In this section, we present some of the mesh definition and manipulation tools provided by Feel.

2.2.1 Mesh definition

We look at the definition of a mesh data structure. First, we define the type of geometric entities that we shall use to form our mesh. Feel supports

- simplices: segment, triangle, tetrahedron
- tensorized entities: segment, quadrangle, hexahedron

We choose between `Simplex<Dim,Order,RealDim>` and `SimplexProduct<Dim,Order,RealDim>`. They have the same template arguments:

- `Dim`: the topological dimension of the entity
- `Order`: the order of the entity(usually 1, higher order in development)
- `RealDim`: the dimension of the real space

```
typedef Simplex<Dim> convex_type;
//typedef Hypercube<Dim, 1,Dim> convex_type;
```

Then we define the mesh type, `Mesh<Entity>` by passing as argument the type of entity it is formed with. At the moment hybrid meshes are not supported.

```
typedef Mesh<convex_type > mesh_type;
typedef boost::shared_ptr<mesh_type> mesh_ptrtype;
```

It is customary, and usually a very good practice, to define the `boost::shared_ptr<>` counterpart which is used actually in practice. We can now instantiate a new mesh data structure.

The next step is to read some mesh files. Feel supports essentially the Gmsh mesh file format. It provides also some classes to manipulate Gmsh `.geo` files and generate `.msh` files. To begin, we use some helper classes to generate a `.geo` file.

- `GmshTensorizedDomain` will allow to create a tensorized domain (e.g. cube) in 1D, 2D and 3D. It allows to modify
  - the characteristic size of the mesh (by default $h = 0.1$)
  - the domain, by default it is the cube $[0;1] \times [0;1] \times [0;1]$
- `GmshSimplexDomain` will allow to create a simplex domain (e.g. segment, triangle or tetrahedron). Again you can modify
  - the characteristic size of the mesh (by default $h = 0.1$)
  - the domain vertices, by default $(-1,-1,-1), (1,-1,-1), (-1,1,-1), (-1,-1,1)$

Here is an example

```
auto mesh = createGMSHMesh( _mesh=new mesh_type,
                            _desc=domain( _name=(boost::format( "%1%-%2%" ) % shape % Dim)
                                          _shape=shape,
                                          _dim=Dim,
                                          _h=X[0] ) );
//#endmarker4#


//# marker62 #
exporter->step(0)->setMesh( mesh );
exporter->save();
```

```
    //# endmarker62 #
}

//
// main function: entry point of the program
//
int main( int argc, char** argv )
{
    Feel::Environment env( argc, argv );

    Application app( argc, argv, makeAbout(), makeOptions() );
    if ( app.vm().count( "help" ) )
    {
        std::cout << app.optionsDescription() << "\n";
        return 0;
    }

    app.add( new MyMesh<1>( app.vm(), app.about() ) );
    app.add( new MyMesh<2>( app.vm(), app.about() ) );
    app.add( new MyMesh<3>( app.vm(), app.about() ) );

    app.run();
}
```

The call to `setCharacteristicLength` allows to change the mesh size to `M_meshSize` which was given for example on the command line using the `Application` framework, see section 2.1. The call to `generate` creates the `.geo` file and generate the associated mesh. Note that `fname` holds the name of the `.msh` file, e.g. `mymesh.msh`. Feel adds automatically the `.msh` extension. Also note the last argument of `GmshTensorizedDomain`, it allows to change the type of geometric entities used to generate the mesh. Here we use `Simplex`, we could have also used `SimplexProduct` (quadrangle or hexahedron).

Next we import the mesh using the `ImporterGmsh` class as follows:

At this stage we are ready to use the mesh instance: we can for example export the mesh to a postprocessing format. Two formats are supported at the moment

- `Ensight` (case and sos) which is supported by the software Ensight[2] and Paraview[3]
- `Gmsh` which is post-processing format of Gmsh

The export stage reads as follows:

In this example, we create a $\mathbb{P}_0$ function space and save the piecewise constant function that associates to each element the process id it belongs to. In sequential, they all belong to processor 0. In a parallel setting, the mesh is partitioned using Metis and each element is associated with a corresponding processor. The figure 2.2.1 displays a partitioned mesh in two regions and the associated $\mathbb{P}_0$ function.

2.2.2 Iterating over the entities of a mesh

Feel mesh data structures provides powerful iterators that allows to walk though the mesh in various ways: iterate over element, faces , points, marked[4] elements, marked faces, ...

2.3 Function Spaces

2.3.1 Defining function spaces and functions

- basis function
- function spaces
- element of a function space

[2] http://www.ensight.com
[3] http://www.paraview.org
[4] associated to an integer flag denoting a region, material, processor

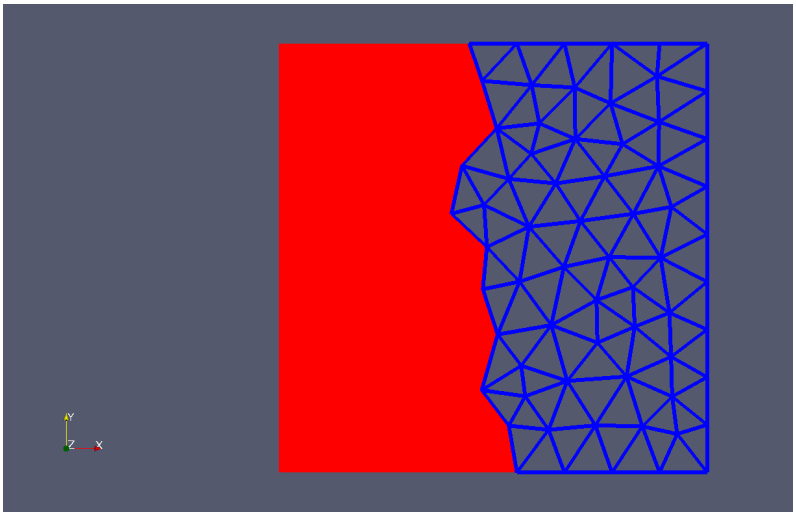


Figure 2.1: Screenshot of Paraview (3.2.1) of a 2D mesh partitioned and distributed on two processors

2.3.2 Using functions spaces and functions

- interpolating
- nodal projection
- saving

2.4 Linear Algebra

Feel supports three different linear algebra environments that we shall call *backends*.

- Gmm[5]
- Petsc[6]
- Trilinos[7]

2.4.1 Choosing a linear algebra backend

To select a backend in order to solve a linear system, we instantiate the `Backend` class associated.

```
#include <feel/feelalg/backend.hpp>
boost::shared_ptr<Backend<double> > backend =
     Backend<double>::build( BACKEND_PETSC );
```

The backend provides an interface to solve

$$Ax = b \tag{2.1}$$

where A is a $n \times n$ sparse matrix and x, b vectors of size n. The backend defines the C++ types for each of these, e.g.

```
Backend<double>::sparse_matrix_type A;
Backend<double>::vector_type x,b;
```

5
6
7

In practice, we use the `boost::shared_ptr<>` shared pointer to ensure that we won't get memory leaks. The backends provide a corresponding **typedef**

```
Backend<double>::sparse_matrix_ptrtype A( backend->newMatrix( Xh, Yh ) );
Backend<double>::vector_ptrtype x( backend->newVector( Yh ) );
Backend<double>::vector_ptrtype b( backend->newVector( Xh ) );
```

where X_h and Y_h are function spaces providing the number of degrees of freedom that will define the size of the matrix and vectors thanks to the helpers functions `Backend::newMatrix()` and `Backend::newVector`. In a parallel setting, the local/global processor mapping would be passed down by the function spaces.

2.4.2 Defining and using matrices and vectors

2.4.3 Solving

2.5 Variational Formulation

- keywords
- principles

2.5.1 Computing integrals

myintegrals.cpp

We would like to compute some integrals on a domain of $\Omega = [0,1]^d \subset \mathbb{R}^d$ and parts of the domain, i.e. subregions and (parts of) boundary.

Once we have defined the computational mesh, we would like to compute the area of the domain. We form the integral $\int_\Omega 1$, the code reads as follows

```
    double local_domain_area = integrate( elements(mesh),
                                          constant(1.0)).evaluate()(0,0);
```

`elements(mesh)` returns a pair of iterators over the elements owned by the current processor, `im` is an instance of the `im_type` which provides a quadrature method to integrate exactly polynomials up to degree 2. In our case integrating constant(degree 0) would have sufficed, but we will reuse `im` later. Now that we have computed the integral of 1 over the region of Ω current processor (ie the area of the domain owned by the processor), we want to compute the area of Ω. To do that we collect the integrals on all processors using a `reduce` MPI operation and sum all contributions. We have used here the Boost.MPI library that provides an extremely powerful C++ wrapper around the MPI library. The code reads

```
    double global_domain_area=local_domain_area;
    if ( this->comm().size()  > 1 )
        mpi::all_reduce( this->comm(),
                         local_domain_area,
                         global_domain_area,
                         std::plus<double>() );
```

Finally, we print to the log file the result of the local and global integral calculation. Another calculation is for example to compute the perimeter of the domain

```
    Log() << "int_Omega 1 = " << global_domain_area
          << "[ " << local_domain_area << " ]\n";
```

the main difference with the domain area computation resides in the elements with are iterating on: here we are iterating on the boundary faces of the domain to compute the integral using `boundaryfaces(mesh)` to provide the pairs of iterators.

Now say that we want to compute

$$\int_\Omega x^2 + y^2 dxdy. \tag{2.2}$$

The Finite Element Embedded Language (FEEL++) language provides the keyword `Px()` and `Py()` to denote the x and y coordinates like in equation (2.2). The code reads then

```
        double local_boundary_length = integrate( boundaryfaces(mesh),
                                                  constant(1.0)).evaluate()(0,0);
        double global_boundary_length = local_boundary_length;
        if ( this->comm().size() > 1 )
            mpi::all_reduce( this->comm(),
                             local_boundary_length,
                             global_boundary_length,
                             std::plus<double>() );
        Log() << "int_BoundaryOmega (1)= " << global_boundary_length
              << "[ " << local_boundary_length << " ]\n";
```

Note that in this case, we really require the use of a quadrature that integrates exactly order 2 polynomials.

Let's run now the tutrial example `myintegrals`. The results are stored in the log file under `~/feel/myintegrals/`.

```
> cat ~/feel/myintegrals/Simplex_2_1/h_0.5/myintegrals-1.0
myintegrals-1.0 is opened for debug
[Area 0] int_Omega = 1[ 1 ]
[Area 0] int_Omega = 4[ 4 ]
[Area 0] int_Omega = 0.666667[ 0.666667 ]
```

We remark that the results are exact. Integrating higher order polynomials (≥ 3) or non-polynomial function would typically require higher order quadrature to get accurate results. To do that increase `imOrder` in the example and try integrating $f(x,y) = x^3 + xy^2$.

In order to see what happens in parallel, use `mpirun` to launch `myintegrals` on several processors, for example

```
> mpirun -np 4 myintegrals --hsize=0.1
> cat ~/feel/myintegrals/Simplex_2_1/h_0.1/myintegrals-4.0
myintegrals-4.0 is opened for debug
[Area 0] int_Omega = 1[ 0.253348 ]
[Area 0] int_Omega = 4[ 1.44444 ]
[Area 0] int_Omega = 0.666667[ 0.0701812 ]
> cat ~/feel/myintegrals/Simplex_2_1/h_0.1/myintegrals-4.1
myintegrals-4.1 is opened for debug
[Area 0] int_Omega = 1[ 0.288919 ]
[Area 0] int_Omega = 4[ 0.444444 ]
[Area 0] int_Omega = 0.666667[ 0.186251 ]
> cat ~/feel/myintegrals/Simplex_2_1/h_0.1/myintegrals-4.2
myintegrals-4.2 is opened for debug
[Area 0] int_Omega = 1[ 0.183219 ]
[Area 0] int_Omega = 4[ 1.11111 ]
[Area 0] int_Omega = 0.666667[ 0.105008 ]
> cat ~/feel/myintegrals/Simplex_2_1/h_0.1/myintegrals-4.3
myintegrals-4.3 is opened for debug
[Area 0] int_Omega = 1[ 0.274514 ]
[Area 0] int_Omega = 4[ 1 ]
[Area 0] int_Omega = 0.666667[ 0.305227 ]
```

2.5.2 Standard formulation: the Laplacian case

Mathematical formulation

`laplacian.cpp`

In this example, we would like to solve for the following problem in 2D

Problem 1 *find u such that*

$$-\Delta u = f \text{ in } \Omega = [-1;1]^2 \tag{2.3}$$

with

$$f = 2\pi^2 g \tag{2.4}$$

and g is the exact solution

$$g = \sin(\pi x)\cos(\pi y) \tag{2.5}$$

The following boundary conditions apply

$$u = g_{|x=\pm 1}, \quad \frac{\partial u}{\partial n} = 0_{|y=\pm 1} \tag{2.6}$$

We propose here two possible variational formulations. The first one, handles the Dirichlet boundary conditions strongly, that is to say the condition is *incorporated* into the function space definitions. The second one handles the Dirichlet condition *weakly* and hence we have a uniform treatment for all types of boundary conditions.

Strong Dirichlet conditions The variational formulation reads as follows, we introduce the spaces

$$\mathcal{X} = \left\{ v \in H_1(\Omega) \text{ such that } v = g_{|x=-1,x=1} \right\} \tag{2.7}$$

and

$$\mathcal{V} = \left\{ v \in H_1(\Omega) \text{ such that } v = 0_{|x=-1,x=1} \right\} \tag{2.8}$$

We multiply (2.3) by $v \in \mathcal{V}$ then integrate over Ω and obtain

$$\int_\Omega -\Delta u v = \int_\Omega f v \tag{2.9}$$

We integrate by parts and reformulate the problem as follows:

Problem 2 *we look for $u \in \mathcal{X}$ such that for all $v \in \mathcal{V}$*

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v \tag{2.10}$$

In the present space setting (2.8) and boundary conditions (2.6), we have the boundary term from the integration by parts which is dropped being equal to 0

$$\int_{\partial\Omega} \frac{\partial u}{\partial n} v = 0, \tag{2.11}$$

recalling that

$$\frac{\partial u}{\partial n} \overset{\text{def}}{=} \nabla u \cdot n \tag{2.12}$$

where n is the outward normal to $\partial\Omega$ by convention.We now discretize the problem, we create a mesh out of Ω, we have

$$\Omega = \cup_{e=1}^{N_{\text{el}}} \Omega^e \tag{2.13}$$

where Ω^e can be segments, triangles or tetrahedra depending on d and we have N_{el} of them. We introduce the finite dimensional spaces of continuous piecewise polynomial of degree N functions

$$X_h = \left\{ v_h \in C^0(\Omega),\ v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e),\ v_h = g_{|x=-1,x=1} \right\} \tag{2.14}$$

and

$$V_h = \left\{ v_h \in C^0(\Omega),\ v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e),\ v_h = 0_{|x=-1,x=1} \right\} \tag{2.15}$$

which are out trial and test function spaces respectively. We now have the problem we seek to solve which reads in our continuous Galerkin framework

Problem 3 *we look for $u_h \in X_h \subset \mathcal{X}$ such that for all $v \in V_h \subset \mathcal{V}$*

$$\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f v_h \tag{2.16}$$

Weak Dirichlet conditions There is an alternative formulation which allows to treat weakly Dirichlet(Essential) boundary conditions similarly to Neumann(Natural) and Robin conditions. Following a similar development as in the previous section, the problem reads

Problem 4 *we look for* $u \in X_h \subset H_1(\Omega)$ *such that for all* $v \in X_h$

$$\int_\Omega \nabla u \cdot \nabla v + \int_{|x=-1,x=1} -\frac{\partial u}{\partial n} v - u\frac{\partial v}{\partial n} + \frac{\mu}{h} uv = \int_\Omega fv + \int_{|x=-1,x=1} -g\frac{\partial v}{\partial n} + \frac{\mu}{h} gv \tag{2.17}$$

where

$$X_h = \left\{ v_h \in C^0(\Omega),\ v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e) \right\} \tag{2.18}$$

In (3.5), g is defined by (2.5). μ serves as a penalisation parameter which should be > 0, e.g. between 2 and 10, and h is the size of the face. The inconvenient of this formulation is the introduction of the parameter μ, but the advantage is the *weak* treatment of the Dirichlet condition.

Feel formulation

First we define the f and g. To do that we use the `AUTO` keyword and associate to `f` and `g` their expressions

```
value_type pi = M_PI;
//! deduce from expression the type of g (thanks to keyword 'auto')
auto g = sin(pi*Px())*cos(pi*Py())*cos(pi*Pz());
gproj = vf::project( Xh, elements(mesh), g );

//! deduce from expression the type of f (thanks to keyword 'auto')
auto f = pi*pi*Dim*g;
```

where `M_PI` is defined in the header `cmath`. Using `AUTO` allows to defined `f` and `g` — which are moderately complex object — without having to know the actual type. `AUTO` determines automatically the type of the expression using the `__typeof__` keyword internally.

Then we form the right hand side by defining a linear form whose algebraic representation will be stored in a `vector_ptrtype` which is provided by the chosen linear algebra backend. The linear form is equated with an integral expression defining our right hand side.

```
vector_ptrtype F( M_backend->newVector( Xh ) );
form1( _test=Xh, _vector=F, _init=true ) =
    integrate( elements(mesh), f*id(v) )+
    integrate( markedfaces( mesh, mesh->markerName("Neumann") ), nu*gradv(gproj)*vf::N()*i
```

`form1` generates an instance of the object representing linear forms, that is to say it mimics the mathematical object ℓ such that

$$\begin{array}{rccc} \ell: & X_h & \mapsto & \mathbb{R} \\ & v_h & \to & \ell(v_h) = \int_\Omega fv \end{array} \tag{2.19}$$

which is represented algebraically in the code by the vector `F` using the argument `_vector`. The last argument `_init`, if set to **`true`**[8], will zero-out the entries of the vector `F`.

We now turn to the left hand side and define the bilinear form using the `form2` helper function which is passed *(i)* the trial function space using the `_trial` option, *(ii)* the test function space using the `_test` option, *(iii)* the algebraic representation using `_matrix`, i.e. a sparse matrix whose type is derived from one of the linear algebra backends and *(iv)* whether the associated matrix should initialized using `_init`.

```
/** \code */
sparse_matrix_ptrtype D( M_backend->newMatrix( Xh, Xh ) );
/** \endcode */

//! assemble ∫_Ω ν∇u·∇v
/** \code */
form2( Xh, Xh, D, _init=true ) =
    integrate( elements(mesh), nu*gradt(u)*trans(grad(v)) );
/** \endcode */
```

[8] It is set to **`false`** by default.

Finally, we deal with the boundary condition, we implement both formulation described in appendix **??**. For a *strong* treatment of the Dirichlet condition, we use the `on()` keyword of FEEL++ as follows

```
D->close();
form2( Xh, Xh, D ) +=
    on( markedfaces(mesh, mesh->markerName("Dirichlet")), u, F, g );
```

Notice that we add, using `+=`, the Dirichlet contribution for the bilinear form. The first argument is the set of boundary faces to apply the condition: in gmsh the points satisfying $x = \pm 1$ are marked using the flags 1 and 3 ($x = -1$ and $x = 1$ respectively.)

To implement the weak Dirichlet boundary condition, we add the following contributions to the left and right hand side:

```
form1( _test=Xh, _vector=F ) +=
    integrate( markedfaces(mesh,mesh->markerName("Dirichlet")), g*(-grad(v)*vf::N(
```

Note that we use the command line option `--weakdir` set to 1 by default to decide between weak/strong Dirichlet handling. Apart the uniform treatment of boundary conditions, the weak Dirichlet formulation has the advantage to work also in a parallel environment.

Next we solve the linear system

$$Du = F \tag{2.20}$$

where the `solve` function is implemented as follows

Finally we check for the L_2 error in our approximation by computing

$$\|u - u_h\|_{L_2} = \sqrt{\int_\Omega (u - u_h)^2} = \sqrt{\int_\Omega (g - u_h)^2} \tag{2.21}$$

where u is the exact solution and is equal to g and u_h is the numerical solution of the problem (2.3) and the components of u_h in the P_2 Lagrange basis are given by solving (2.20).

The code reads

```
double L2error2 =integrate(elements(mesh),
                           (idv(u)-g)*(idv(u)-g) ).evaluate()(0,0);
double L2error =   math::sqrt( L2error2 );


Log() << "||error||_L2=" << L2error << "\n";
```

You can now verify that the L_2 error norm behaves like $h^{-(N+1)}$ where h is the mesh size and N the polynomial order. The H_1 error norm would be checked similarly in h^{-N}. The figure 2.3 displays the results using Paraview.

2.5.3 Mixed formulation: the Stokes case

`stokes.cpp`

Mathematical formulation

We are now interested in solving the Stokes equations, we would like to solve for the following problem in 2D

Problem 5 *find* $(\mathbf{u}, p)$ *such that*

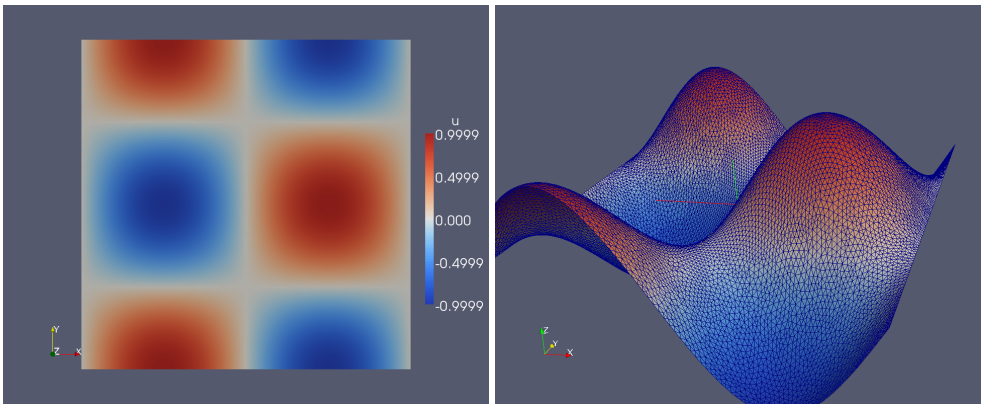
$$-\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \quad \textit{and} \quad \nabla \cdot \mathbf{u} = 0, \quad \textit{in } \Omega = [-1;1]^2 \tag{2.22}$$

with

$$\mathbf{f} = \mathbf{0} \tag{2.23}$$

where μ *being the viscosity. The following boundary conditions apply*

$$\mathbf{u} = \mathbf{1}_{|y=1}, \quad \mathbf{u} = \mathbf{0}_{|\partial\Omega\setminus\{(x,y)\in\Omega|y=1\}} \tag{2.24}$$



(a) Colored with u (b) Elevation

Figure 2.2: Solution of problem 4

In problem (3), p is known up to a constant c, *i.e.* if p is a solution then $p + c$ is also solution. To ensure uniqueness we impose the constraint that p should have zero-mean, *i.e.*

$$\int_\Omega p = 0 \tag{2.25}$$

The problem 5 now reads

Problem 6 *find* $(\mathbf{u}, p, \lambda)$ *such that*

$$-\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \quad , \quad \nabla\cdot\mathbf{u} + \lambda = 0, \quad \textit{and} \quad \int_\Omega p = 0, \quad \textit{in } \Omega = [-1;1]^2 \tag{2.26}$$

with

$$\mathbf{f} = \mathbf{0} \tag{2.27}$$

where μ being the viscosity. The following boundary conditions apply

$$\mathbf{u} = \mathbf{1}_{|y=1}, \quad \mathbf{u} = \mathbf{0}_{|\partial\Omega\setminus\{(x,y)\in\Omega|y=1\}} \tag{2.28}$$

The functional framework is as follows, we look for $\mathbf{u}$ is $H^1_0(\Omega)$ and p in $L^2_0(\Omega)$. We shall not seek p in $L^2_0(\Omega)$ but rather in $L^2(\Omega)$ and use Lagrange multipliers which live are the constants whose space we denote $\mathbb{P}_0(\Omega)$, to enforce (2.25).

Denote $\mathcal{X} = H^1_0(\Omega)\times L^2(\Omega)\times\mathbb{P}_0(\Omega)$, the variational formulation reads we look for $(\mathbf{u}, p, \lambda) \in \mathcal{X}$ for all $(\mathbf{v}, q, \nu) \in \mathcal{X}$

$$\int_\Omega \mu\nabla\mathbf{u} : \nabla\mathbf{v} + \nabla\cdot\mathbf{v}p + \nabla\cdot\mathbf{u}\, q + q\lambda + p\nu \;=\; \int_\Omega \mathbf{f}\cdot\mathbf{v} \tag{2.29}$$

We build a triangulation Ω_h of Ω, we choose compatible (piecewise polynomial) discretisation spaces X_h and M_h, *e.g.* the Taylor Hood element ($\mathbb{P}_N/\mathbb{P}_{N-1}$) and we denote $\mathcal{X}_h = X_h \times M_h \times \mathbb{P}_0(\Omega)$. The discrete problem now reads, we look for $(\mathbf{u}_h, p_h, \lambda_h) \in \mathcal{X}_h$ such that for all $(\mathbf{v}_h, q_h, \nu_h) \in \mathcal{X}_h$

$$\int_{\Omega_h} \mu\nabla\mathbf{u}_h\cdot\nabla\mathbf{v}_h + \nabla\cdot\mathbf{v}_h\, p_h + \nabla\cdot\mathbf{u}_h\, q_h + p_h\nu_h + q_h\lambda_h = \int_{\Omega_h}\mathbf{f}\cdot\mathbf{v}_h \tag{2.30}$$

The formulation (2.30) leads to a linear system of the form

$$\underbrace{\begin{pmatrix} A & B & 0\\ B^T & 0 & C\\ 0 & C^T & 0\end{pmatrix}}_{\mathcal{A}}\underbrace{\begin{pmatrix}\mathbf{u}_h\\ p_h\\ \lambda_h\end{pmatrix}}_{\mathcal{U}} = \underbrace{\begin{pmatrix}F\\0\\0\end{pmatrix}}_{\mathcal{F}} \tag{2.31}$$

where A corresponds to the $(\mathbf{u}, \mathbf{v})$ block, B to the $(\mathbf{u}, q)$ block and C to the (p, ν) block. $\mathcal{A}$ is a symetric positive definite matrix and thus the system $\mathcal{A}\mathcal{U} = \mathcal{F}$ enjoys a unique solution.

Feel formulation

Regarding the implementation of the Stokes problem 5, we can start from the laplacian case, from section 2.5.2. The implementation we choose to display here defines and builds $\mathcal{X}_h$, $\mathcal{A}, \mathcal{U}$ and $\mathcal{F}$.

We start by defining and building $\mathcal{X}_h$: first we define the basis functions that will span each subspaces X_h, M_h and $\mathbb{P}_0(\Omega)$.

```
typedef BasisU basis_u_type;
typedef BasisP basis_p_type;
typedef Lagrange<0, Scalar> basis_l_type;
typedef bases<basis_u_type,basis_p_type, basis_l_type> basis_type;
```

note that on the **typedef** we build a (MPL) vector of them. Now we are ready to define the function-space $\mathcal{X}_h$, much like in the Laplacian case:

```
typedef FunctionSpace<mesh_type, basis_type> space_type;
typedef boost::shared_ptr<space_type> space_ptrtype;
```

Next we define a few types which are associated with $\mathcal{U}$, u, p and λ respectively.

```
typedef typename space_type::element_type element_type;
typedef typename element_type::template sub_element<0>::type element_0_type;
typedef typename element_type::template sub_element<1>::type element_1_type;
typedef typename element_type::template sub_element<2>::type element_2_type;
```

Using these types we can instantiate elements of $\mathcal{X}_h$, X_h, M_h and $\mathbb{P}_0(\Omega_h)$ respectively:

```
space_ptrtype Xh = space_type::New( mesh );

auto U = Xh->element();
auto V = Xh->element();
auto u = U.template element<0>();
auto v = V.template element<0>();
auto p = U.template element<1>();
auto q = V.template element<1>();

auto lambda = U.template element<2>();
auto nu = V.template element<2>();
```

They will serve in the definition of the variational formulation. We can now start assemble the various terms of the variational formulation (2.30). First we define some viscous stress tensor, $\tau(\mathbf{u}) = \nabla\mathbf{u}$, associated with the trial and test functions respectively

```
auto deft = gradt(u);
auto def = grad(v);
```

Then we define the total stress tensor times the normal, $\bar{\sigma}(\mathbf{u}, p)\mathbf{n} = -p\mathbf{n} + 2\mu\tau(\mathbf{u})\mathbf{n}$ where $\mathbf{n}$ is the normal and $\bar{\sigma}(\mathbf{u}, p) = -p\mathbb{I} + 2\mu\tau(\mathbf{u})$:

```
// total stress tensor (trial)
auto SigmaNt = -idt(p)*N()+mu*deft*N();

// total stress tensor (test)
auto SigmaN = -id(p)*N()+mu*def*N();
```

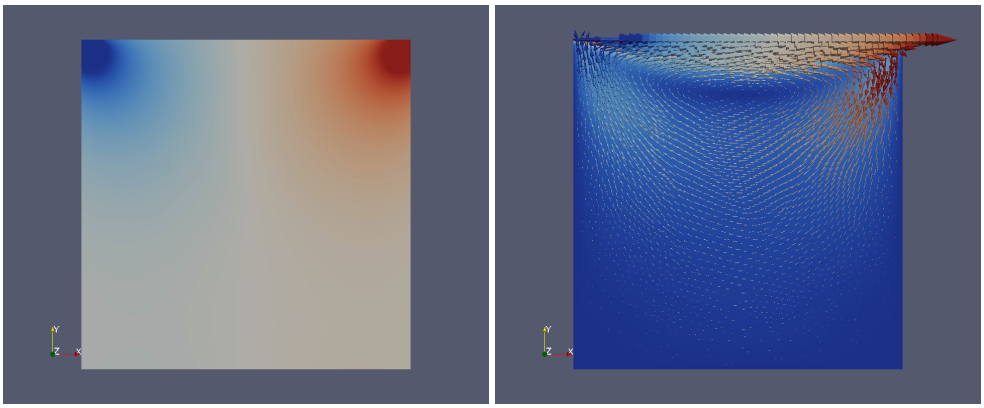
We then form the matrix $\mathcal{A}$ starting with block A, block B block C and finally the boundary conditions.

```
auto D =  M_backend->newMatrix( Xh, Xh );

form2( Xh, Xh, D, _init=true )=integrate( elements(mesh), mu*trace(deft*trans(def)) );
form2( Xh, Xh, D )+=integrate( elements(mesh), - div(v)*idt(p) + divt(u)*id(q) );
form2( Xh, Xh, D )+=integrate( elements(mesh), id(q)*idt(lambda) + idt(p)*id(nu) );
form2( Xh, Xh, D )+=integrate( boundaryfaces(mesh), -trans(SigmaNt)*id(v) );
form2( Xh, Xh, D )+=integrate( boundaryfaces(mesh), -trans(SigmaN)*idt(u) );
form2( Xh, Xh, D )+=integrate( boundaryfaces(mesh), +penalbc*trans(idt(u))*id(v)/hFace() );
```

The figure 2.3 displays p and $\mathbf{u}$ which are available in

```
ls ~/feel/doc/tutorial/stokes/Simplex_2_1_2/P2/h_0.05
```



(a) Colored with p, $h = 0.05$

(b) Colored with $\|\mathbf{u}\|$ and the arrows associated to $\mathbf{u}$ colored with p

Figure 2.3: Solution of problem 5

Chapter 3

Examples

3.1 Solving nonlinear equations

Feel allows to solve nonlinear equations thanks to its interface to the interface to the PETSc nonlinear solver library. It requires the implementation of two extra functions in your application that will update the jacobian matrix associated to the tangent problem and the residual.

Consider that you have an application class `MyApp` with a backend as data member

```
#include <feel/feelcore/feel.hpp>
#include <feel/feelcore/application.hpp>
#include <feel/feelalg/backend.hpp>
namespace Feel {

class MyApp : public Application
{
  public:

  typedef Backend<double> backend_type;
  typedef boost::shared_ptr<backend_type> backend_ptrtype;

  MyApp( int argc, char** argv,
  AboutData const& ad, po::options_description const& od )
  :
  // init the parent class
  Application( argc, argv, ad, od ),
  // init the backend
  M_backend( backend_type::build( this->vm() ) ),
  {
    // define the callback functions (works only for the PETSc backend)
    M_backend->nlSolver()->residual =
      boost::bind( &self_type::updateResidual, boost::ref( *this ), _1, _2 );
    M_backend->nlSolver()->jacobian =
      boost::bind( &self_type::updateJacobian, boost::ref( *this ), _1, _2 );

  }
  void updateResidual( const vector_ptrtype& X, vector_ptrtype& R )
  {
    // update the matrix J (Jacobian matrix) associated
    // with the tangent problem
  }
  void updateJacobian( const vector_ptrtype& X, sparse_matrix_ptrtype& J)
  {
    // update the vector R associated with the residual
  }
  void run()
  {

    //define space
    Xh...
    element_type u(Xh);
    // initial guess is 0
```

```
    u = project( M_Xh, elements(mesh), constant(0.) );
    vector_ptrtype U( M_backend->newVector( u.functionSpace() ) );
    *U = u;

    // define R and J
    vector_ptrtype R( M_backend->newVector( u.functionSpace() ) );
    sparse_matrix_ptrtype J;

    // update R
    updateJacobian( U, R );
    // update J
    updateResidual( U, J );

    // solve using non linear methods (newton)
    // tolerance : 1e-10
    // max number of iterations : 10
    M_backend->nlSolve( J, U, R, 1e-10, 10 );

    // the soluution was stored in U
    u = *U;
  }
  private:

  backend_ptrtype M_backend;
};
} // namespace Feel
```

The function `updateJacobian` and `updateResidual` implement the assmebly of the matrix J (jacobian matrix) and the vector R (residual vector) respectively.

3.1.1 A first nonlinear problem

As a simple example, let Ω be a subset of $\mathbb{R}^d, d = 1, 2, 3$, (*i.e.* $\Omega = [-1, 1]^d$) with boundary $\partial\Omega$. Consider now the following equation and boundary condition

$$-\Delta u + u^\lambda = f, \quad u = 0 \text{ on } \partial\Omega. \tag{3.1}$$

where $\lambda \in \mathbb{R}_+$ is a given parameter and $f = 1$.

> To be described in this section. For now see `doc/tutorial/nonlinearpow.cpp` for an implementation of this problem.

3.1.2 Simplified combustion problem: Bratu

As a simple example, let Ω be a subset of $\mathbb{R}^d, d = 1, 2, 3$, (*i.e.* $\Omega = [-1, 1]^d$) with boundary $\partial\Omega$. Consider now the following equation and boundary condition

$$-\Delta u + \lambda e^u = f, \quad u = 0 \text{ on } \partial\Omega \tag{3.2}$$

where λ is a given parameter. Ceci est généralement appellé le problème de Bratu et apparaît lors de la simplification de modèles de processus de diffusion non-linéaires par exemple dans le domaine de la combustion.

> To be described in this section. For now see `doc/tutorial/bratu.cpp` for an implementation of this problem.

3.2 Natural convection in a heated tank

3.2.1 Description

The goal of this project is to simulate the fluid flow under natural convection: the heated fluid circulates towards the low temperature under the action of density and gravity differences. Thie phenomenon is

important in the sense it models evacuation of heat, generated by friction forces for example, with a cooling fluid.

We shall put in place a simple convection problem in order to study the phenomenon without having to handle the difficulties of more complex domaines. We describe then some necessary transformations to the equations, then we define quantities of interest to be able to compare the simulations with different parameter values.

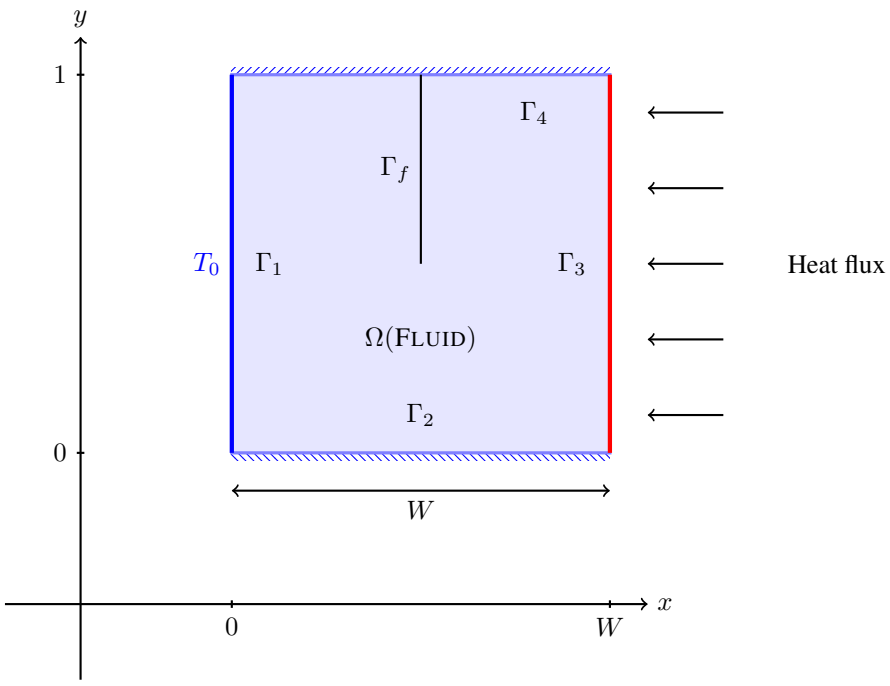


Figure 3.1: Geometry of the model

To study the convection, we use a model problem: it consists in a rectangular tank of height 1 and width W, in which the fluid is enclosed, see figure 3.1. We wish to know the fluid velocity $\mathbf{u}$, the fluid pressure p and fluid temperature θ.

We introduce the adimensionalized Navier-Stokes and heat equations parametrized by the Grashof and Prandtl numbers. These parameters allow to describe the various regimes of the fluid flow and heat transfer in the tank when varying them.

The adimensionalized steady incompressible Navier-Stokes equations reads:

$$\begin{aligned} \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p - \frac{1}{\sqrt{\mathrm{Gr}}}\Delta\mathbf{u} &= \theta\mathbf{e}_2 \\ \nabla\cdot\mathbf{u} &= 0 \text{ sur } \Omega \\ \mathbf{u} &= \mathbf{0} \text{ sur } \partial\Omega \end{aligned} \tag{3.3}$$

where Gr is the Grashof number, $\mathbf{u}$ the adimensionalized velocity and p adimensionalized pressure and θ the adimensionalized temperature. The temperature is in fact the difference between the temperature in the tank and the temperature T_0 on boundary Γ_1.

The heat equation reads:

$$\begin{aligned} \mathbf{u}\cdot\nabla\theta - \frac{1}{\sqrt{\mathrm{Gr}}\mathrm{Pr}}\Delta\theta &= 0 \\ \theta &= 0 \text{ sur } \Gamma_1 \\ \frac{\partial\theta}{\partial n} &= 0 \text{ sur } \Gamma_{2,4} \\ \frac{\partial\theta}{\partial n} &= 1 \text{ sur } \Gamma_3 \end{aligned} \tag{3.4}$$

where Pr is the Prandtl number.

3.2.2 Influence of parameters

what are the effects of the Grashof and Prandtl numbers ? We remark that both terms with these parameters appear in front of the Δ parameter, they thus act on the diffusive terms. If we increase the Grashof number or the Prandtl number the coefficients multiplying the diffusive terms decrease, and this the convection, that is to say the transport of the heat via the fluid, becomes dominant. This leads also to a more difficult and complex flows to simulate, see figure 3.2. The influence of the Grashof and Prandtl numbers are different but they generate similar difficulties and flow configurations. Thus we look only here at the influence of the Grashof number which shall vary in $[1, 1e7]$.

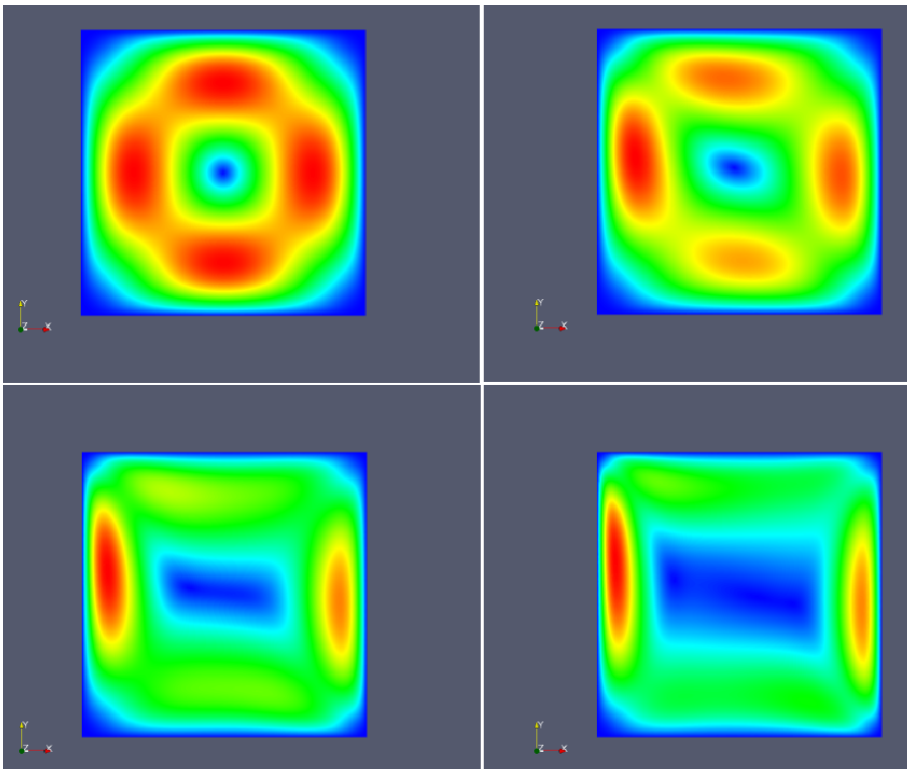


Figure 3.2: Velocity norm with respect to Grashof, $\mathrm{Gr} = 100, 10000, 100000, 500000$. $h = 0.01$ and $\mathrm{Pr} = 1$.

3.2.3 Quantities of interest

We would like to compare the results of many simulations with respect to the Grashof defined in the previous section. We introduce two quantities which will allow us to observe the behavior of the flow and

heat transfer.

Mean temperature

We consider first the mean temperature on boundary Γ_3

$$T_3 = \int_{\Gamma_3} \theta \tag{3.5}$$

This quantity should decrease with increasing Grashof because the fluid flows faster and will transport more heat which will cool down the heated boundary Γ_3. We observe this behavior on the figure 3.3.

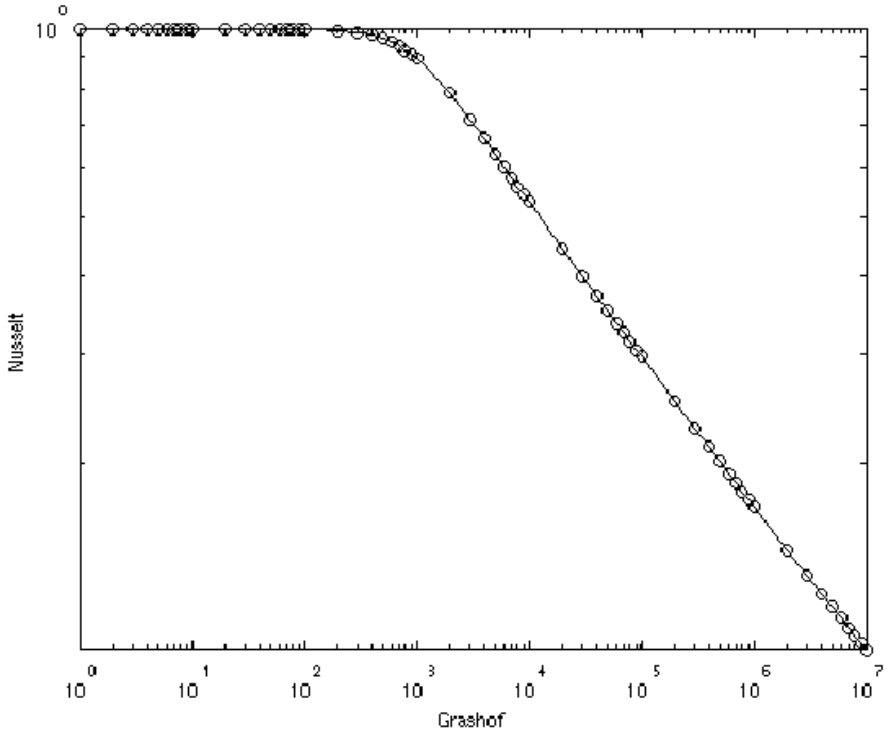


Figure 3.3: Mean temperature with respect to the Grashof number; $h = 0.02$ with $\mathbb{P}_3$ Lagrange element for the velocity, $\mathbb{P}_2$ Lagrange for the pressure and $\mathbb{P}_1$ Lagrange for the temperature.

Flow rate

Another quantity of interest is the flow rate through the middle of the tank. We define a segment Γ_f as being the vertical top semi-segment located at $W/2$ with height $1/2$, see figure 3.1. The flow rate, denoted D_f, reads

$$\mathrm{D}_f = \int_{\Gamma_f} \mathbf{u} \cdot \mathbf{e}_1 \tag{3.6}$$

where $\mathbf{e}_1 = (1, 0)$. Note that the flow rate can be negative or positive depending on the direction in which the fluid flows.

As a function of the Grashof, we shall see a increase in the flow rate. This is true for small Grashof, but starting at $1e3$ the flow rate decreases. The fluid is contained in a boundary layer which is becoming smaller as the Grashof increases.

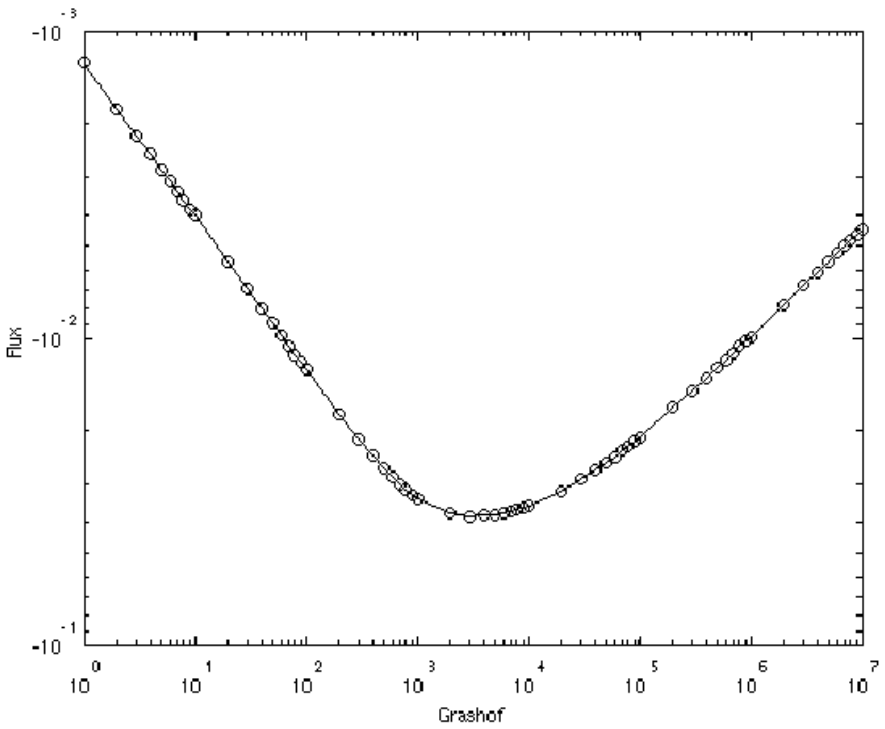


Figure 3.4: Behavior of the flow rate with respect to the Grashof number; $h = 0.02$, $\mathbb{P}_3$ for the velocity, $\mathbb{P}_2$ for the pressure and $\mathbb{P}_1$ for the temperature.

3.2.4 Implementation

This application in implemented in `life/doc/tutorial/convection*.cpp`. The implementation solve the full nonlinear problem using the nonlinear solver framework.

Appendix A

Random notes

A.1 Linear Algebra with PETSC

A.1.1 Using the Petsc Backend: recommended

Using the Petsc backend is recommended. To do that type in the command line

```
myprog --backend=petsc
```

then you can change the type of solvers and preconditioners by adding Petsc options at *the end of the command lines*, for example

```
-pc_type lu
```

will actually solve the problem in one iteration of an iterative solver (p.ex. gmres).

$$PAx = PB \tag{A.1}$$

where $P \approx A^{-1}$. Here A is decomposed in LU form and (A.1) is solved in one iteration.

A.1.2 List of solvers and preconditioners

List of some iterative solvers (Krylov subspace)

- cg, bicg
- gmres, fgmres, lgmres
- bcgs, bcgsl
- see petsc/petscksp.h for more

List of some preconditioners

- lu, choleski
- jacobi, sor
- ilu, icc
- see petsc/petscpc.h for more

A.1.3 What is going on in the solvers?

In order to monitor what is going on (iterations, residual...) Petsc provides some monitoring options

```
-ksp_monitor
```

For example

```
myprog -backend=petsc -ksp_monitor -pc_type lu
```

it should show only one iteration.

See http://www.mcs.anl.gov/petsc/petsc-as/snapshots/petsc-current/docs/manualpages/KSP/KSPMonitorSet.html for more details

A.2 Numerical Schemes

A.2.1 Stokes problem formulation and the pressure

A.2.2 The Stokes problem

Consider the following problem,

$$\text{Stokes:} \left\{ \begin{array}{r} -\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \\ \nabla \cdot \mathbf{u} = 0 \\ \mathbf{u}|_{\partial\Omega} = 0 \end{array} \right. \tag{A.2}$$

where $\Omega \subset \mathbb{R}^d$. There are no boundary condition on the pressure. This problem is ill-posed, indeed we only control the pressure through its gradient ∇p. Thus if $(\mathbf{u}, p)$ is a solution, then $(\mathbf{u}, p + c)$ is also a solution with c any constant. This comes from the way the problem is posed: the box is closed and it is not possible to determine the pressure inside. The remedy is to impose arbitrarily a constraint on the pressure, e.g. its mean value is zero. In other words, we add this new equation to the problem (A.2)

$$\int_\Omega p = 0 \tag{A.3}$$

Remark 1 (The Navier-Stokes case) *This is also true for the incompressible Navier-Stokes equations. We chose Stokes to simplify the exposure.*

A.2.3 Reformulation

In order to impose the condition (A.3), we introduce a new unknown, a Lagrange multiplier, $\lambda \in \mathbb{R}$ and modify the incompressibility equation. Our problem reads now, find $(\mathbf{u}, p, \lambda)$ such that

$$\text{Stokes 2:} \left\{ \begin{array}{rcl} -\mu\Delta\mathbf{u} + \nabla p & = & \mathbf{f} \\ \nabla \cdot \mathbf{u} + \lambda & = & 0 \\ \mathbf{u}|_{\partial\Omega} & = & 0 \\ \int_\Omega p & = & 0 \end{array} \right. \tag{A.4}$$

Remark 2 (The pressure as Lagrange multiplier) *The pressure field p can actually be seen as a Lagrange multiplier for the velocity $\mathbf{u}$ in order to enforce the constraint $\nabla \cdot \mathbf{u} = 0$. λ will play the same role but for the pressure to enforce the condition (A.3). As $h \to 0$, $\lambda \to 0$ as well as the divergence of $\mathbf{u}$. Note also that $\int_\Omega \nabla \cdot \mathbf{u} \approx -\int_\Omega \lambda$ from the second equation.*

A.2.4 Variational formulation

The variational formulation now reads: find $(\mathbf{u}, p, \lambda) \in \mathbf{H}_0^1(\Omega) \times L_0^2(\Omega) \times \mathbb{R}$ such that for all $(\mathbf{v}, q, \eta) \in \mathbf{H}_0^1(\Omega) \times L_0^2(\Omega) \times \mathbb{R}$

$$\text{Stokes 3:} \quad \left\{ \begin{array}{rcl} \int_\Omega \left(\nabla \mathbf{u} \colon \nabla \mathbf{v} + \nabla \cdot \mathbf{v} p \right) & = & \int_\Omega \mathbf{f} \cdot \mathbf{v} \\ \int_\Omega \left(\nabla \cdot \mathbf{u} q + \lambda q \right) & = & 0 \\ \int_\Omega p\eta & = & 0 \end{array} \right. \tag{A.5}$$

Summing up all three equations we get the following condensed formulation:

$$\int_\Omega \nabla \mathbf{u} \colon \nabla \mathbf{v} + \nabla \cdot \mathbf{v} p + \nabla \cdot \mathbf{u} q + \lambda q + \eta p = \int_\Omega \mathbf{f} \cdot \mathbf{v} \tag{A.6}$$

where $\mathbf{H}_0^1(\Omega) = \left\{ \mathbf{v} \in \mathbf{L}^2(\Omega), \nabla \mathbf{v} \in [L^2(\Omega)]^{d \times d},\ \mathbf{v} = 0 \text{ on } \partial\Omega \right\}$, $L_0^2(\Omega) = \left\{ v \in L^2(\Omega),\ \int_\Omega v = 0 \right\}$, and $\mathbf{L}^2(\Omega) = \left\{ \mathbf{v} \in [L^2(\Omega)]^d \right\}$ that is to say each component of a vector field of $\mathbf{L}^2(\Omega)$ are in $L^2(\Omega)$.

A.2.5 Implementation

```
/*basis*/
typedef Lagrange<Order, Vectorial> basis_u_type; // velocity
typedef Lagrange<Order-1, Scalar> basis_p_type; // pressure
typedef Lagrange<0, Scalar> basis_l_type; // multipliers
typedef bases<basis_u_type, basis_p_type, basis_l_type> basis_type;
/*space: product of the velocity, pressure and multiplier spaces*/
typedef FunctionSpace<mesh_type, basis_type, value_type> space_type;
// ...
space_ptrtype Xh = space_type::New( mesh );
element_type U( Xh, "u" );
element_type V( Xh, "v" );
element_0_type u = U.element<0>();
element_0_type v = V.element<0>();
element_1_type p = U.element<1>();
element_1_type q = V.element<1>();
element_2_type lambda = U.element<2>();
element_2_type nu = V.element<2>();
// ...
sparse_matrix_ptrtype D( M_backend->newMatrix( Xh, Xh ) );
form2( Xh, Xh, D, _init=true )=
   integrate( elements(mesh), im,
              // ∇u:∇v
               mu*trace(deft*trans(def))
               // ∇·vp + ∇·uq
               - div(v)*idt(p) + divt(u)*id(q)
               // λq + ηp
               +id(q)*idt(lambda) + idt(p)*id(nu) );
// ...
```

A.2.6 Fix point iteration for Navier-Stokes

Steady incompressible Navier-Stokes equations

Consider the following steady incompressible Navier-Stokes equations, find $(\mathbf{u}, p)$ such that

$$\begin{aligned} \underbrace{\rho \mathbf{u} \cdot \nabla \mathbf{u}}_{\text{convection}} - \underbrace{\nu \Delta \mathbf{u}}_{\text{diffusion}} + \nabla p &= \mathbf{f} \text{ on } \Omega \\ \nabla \cdot \mathbf{u} &= 0 \\ \mathbf{u} &= \mathbf{0} \text{ on } \partial\Omega \end{aligned} \tag{A.7}$$

where ρ is the density of the fluid, ν is the dynamic viscosity of the fluid(la viscosité cinématique $\eta = \nu/\rho$) and $\mathbf{f}$ is the external force density applied to the fluid, (e.g. $\mathbf{f} = -\rho g \mathbf{e}_2$ with $\mathbf{e}_2 = (0, 1)^T$). This equation

system is nonlinear due to the $\mathbf{u} \cdot \nabla \mathbf{u}$ convection term. A simple approach to solve (A.7) is to use a fix point algorithm.

The fixpoint algorithm for NS reads as follows, find $(\mathbf{u}^{(k)}, p^{(k)})$ such that

$$\begin{aligned} \rho \mathbf{u}^{(k-1)} \cdot \nabla \mathbf{u}^{(k)} - \nu \Delta \mathbf{u}^{(k)} + \nabla p^{(k)} &= \mathbf{f} \text{ on } \Omega \\ \nabla \cdot \mathbf{u}^{(k)} &= 0 \\ \mathbf{u}^{(k)} &= 0 \text{ on } \partial\Omega \\ (\mathbf{u}^{(0)}, p^{(0)}) &= (\mathbf{0}, 0) \end{aligned} \tag{A.8}$$

The system (A.8) is now linear at each iteration k and we can write the variational formulation accordingly. A stopping criterium is for example that $\|\mathbf{u}^k - \mathbf{u}^{(k-1)}\| + \|p^k - p^{(k-1)}\| < \epsilon$ where ϵ is a given tolerance (e.g. $1e-4$) and $\|\cdot\|$ is the L_2 norm.

Here is the implementation using Life:

```
// define some tolerance ϵ
epsilon = 1e-4;
// set (u^(0),p^(0)) to (0,0)
velocity_element_type uk(Xh);
velocity_element_type uk1(Xh);
pressure_element_type pk(Ph);
pressure_element_type pk1(Ph);
// by default uk1, uk and pk,pk1 are initialized to 0

// assemble the linear form associated to f
// store in vector F, it does not change over the iterations

// iterations to find (u^(k),p^(k))
do
{
  // save results of previous iterations
  uk1 = uk;
  pk1 = pk;

  //assemble for bilinear form  associated to
  // ρu^(k-1)·∇u^(k) − νΔu^(k) + ∇p^(k)
  // store in matrix A^(k)

  // solve the system A^(k)X = F where X = (u^(k),p^(k))^T

  // use uk,uk1 and pk,pk1 to compute the error estimation at each iteration
  error = ||u^k − u^(k-1)|| + ||p^k − p^(k-1)||
} while( error > epsilon );
```

A.2.7 A Fix point coupling algorithm

Coupling fluid flow and heat transfer: problem

Recall that we have to solve two coupled problems :

$$\text{Heat}(\mathbf{u}) \begin{cases} -\kappa \Delta T + \mathbf{u} \cdot \nabla T &= 0 \\ T|_{\Gamma_1} &= T_0 \\ \frac{\partial T}{\partial \mathbf{n}}|_{\Gamma_3} &= 1 \\ \frac{\partial T}{\partial \mathbf{n}}|_{\Gamma_2, \Gamma_4} &= 0 \end{cases}$$

and

$$\text{Stokes(T)} : \begin{cases} -\nu \Delta \mathbf{u} + \frac{1}{\rho} \nabla p = \mathbf{F} \\ \nabla \cdot \mathbf{u} = 0 \\ \mathbf{u}|_{\partial\Omega} = 0 \end{cases}$$

Where $\mathbf{F}$ can be taken as $\begin{pmatrix} 0 \\ \beta(T - T_0) \end{pmatrix}$ for some $\beta > 0$. β is called the *dilatation coefficient*.

Coupling fluid flow and heat transfer: algorithm

Here is a simple algorithm fix point strategy in pseudo-code:

```
double tol = 1.e-6;
int maxIter = 50;
//Initial guess Un = 0
do
{
  // Find Tn solution of Heat(Un)
  // Find Unp1 solution of Stokes(Tn)
  // compute stopTest = norme(Unp1 - Un)
  // Un = Unp1
}while((stopTest < tol) && (niter <= maxIter));
```

Remark 3 (The unsteady case) *To solve the unsteady problems, one can insert the previous loop in the one dedicated to time discretization*

A.2.8 A Newton coupling algorithm

A fully coupled scheme

Another possiblity is to use a Newton method which allows us to solve the full nonlinear problem coupling velocity, pressure and temperature

$$\text{Find } X \text{ such that } F(X) = 0 \tag{A.9}$$

the method is iterative and reads, find $X^{(n+1)}$ such that

$$J_F(X^{(n)})(X^{(n+1)} - X^{(n)}) = -F(X^{(n)}) \tag{A.10}$$

starting with $X^{(0)} = \mathbf{0}$ or some other initial value and where J_F is the jacobian matrix of F evaluated at $X = ((u_i)_i, (p_i)_i, (\theta_i)_i)^T$. For any ϕ_k, ψ_l and ρ_m the *test* functions associated respectively to velocity, pressure and temperature, our full system reads, Find $X = ((u_i)_i, (p_i)_i, (\theta_i)_i)^T$ such that

$$\begin{array}{lll} F_1((u_i)_i, (p_i)_i, (\theta_i)_i) & = \sum_{i,j} u_i u_j a(\phi_i, \phi_k, \phi_j) - \sum_i p_i b(\phi_k, \psi_i) + \sum_i \theta_i c(\rho_i, \phi_k) + \sum_i u_i d(\phi_i, \phi_k) & = 0 \\ F_2((u_i)_i, (p_i)_i, (\theta_i)_i) & = \sum_i u_i b(\phi_i, \psi_l) & = 0 \\ F_3((u_i)_i, (p_i)_i, (\theta_i)_i) & = \sum_{i,j} u_i \theta_j e(\phi_i, \rho_j, \rho_m) + \sum_i \theta_i f(\rho_i, \rho_m) - g(\rho_m) & = 0 \end{array} \tag{A.11}$$

where $F = (F_1, F_2, F_3)^T$ and

$$\begin{array}{rl} a(\mathbf{u}, \mathbf{v}, \beta) & = \int_\Omega \mathbf{v}^T((\nabla \mathbf{u})\beta) \\ b(\mathbf{v}, p) & = \int_\Omega p(\nabla \cdot \mathbf{v}) - \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} p \\ c(\theta, \mathbf{v}) & = \int_\Omega \theta \mathbf{e}_2 \cdot \mathbf{v} \\ d(\mathbf{u}, \mathbf{v}) & = \frac{1}{\sqrt{\mathrm{Gr}}} \Big(\int_\Omega \nabla \mathbf{u} : (\nabla \mathbf{v})^T - \int_{\partial\Omega} ((\nabla \mathbf{u})\mathbf{n}) \cdot \mathbf{v} \Big) \\ e(\mathbf{u}, \theta, \chi) & = \int_\Omega (\mathbf{u} \cdot \nabla \theta)\chi \\ f(\theta, \chi) & = \frac{1}{\sqrt{\mathrm{GrPr}}} \Big(\int_\Omega \nabla \theta \cdot \nabla \chi - \int_{\Gamma_1} (\nabla \theta \cdot \mathbf{n})\chi \Big) \\ g(\chi) & = \frac{1}{\sqrt{\mathrm{GrPr}}} \int_{\Gamma_3} \chi \end{array} \tag{A.12}$$

Remark 4 *Note that the boundary integrals are kept in order to apply the weak Dirichlet boundary condition trick, see next section A.3.*

Jacobian matrix

In order to apply the newton scheme, we need to compute the jacobian matrix J_F by deriving each equation with respect to each unknowns, ie u_i, p_i and θ_i. Consider the first equation

- Deriving the first equation with respect to u_i we get

$$\frac{\partial F_1}{\partial u_i} = \sum_j u_j a(\phi_i, \phi_k, \phi_j) + \sum_i u_i a(\phi_i, \phi_k, \phi_j) + d(\phi_i, \phi_k) \tag{A.13}$$

- Deriving the first equation with respect to p_i we get

$$\frac{\partial F_1}{\partial p_i} = -b(\phi_k, \psi_l) \tag{A.14}$$

- Deriving the first equation with respect to θ_i we get

$$\frac{\partial F_1}{\partial \theta_i} = c(\rho_i, \rho_k) \tag{A.15}$$

Consider the second equation, only the derivative with respect to u_i is non zero.

$$\frac{\partial F_2}{\partial u_i} = b(\phi_i, \psi_l) \tag{A.16}$$

Finally the third component

- Deriving with respect to u_i

$$\frac{\partial F_3}{\partial u_i} = \sum_j \theta_j e(\phi_i, \rho_j, \rho_m) \tag{A.17}$$

- Deriving with respect to p_i,

$$\frac{\partial F_3}{\partial p_i} = 0 \tag{A.18}$$

- Deriving with respect to $theta_i$,

$$\frac{\partial F_3}{\partial \theta_i} = \sum_j u_j e(\phi_j, \rho_i, \rho_m) + f(\rho_i, \rho_m) \tag{A.19}$$

$$J_F = \begin{pmatrix} \frac{\partial F_1}{\partial u_i} & \frac{\partial F_1}{\partial p_i} & \frac{\partial F_1}{\partial \theta_i} \\ \frac{\partial F_2}{\partial u_i} & \frac{\partial F_2}{\partial p_i}(=0) & \frac{\partial F_2}{\partial \theta_i}(=0) \\ \frac{\partial F_3}{\partial u_i} & \frac{\partial F_3}{\partial p_i}(=0) & \frac{\partial F_3}{\partial \theta_i} \end{pmatrix} \tag{A.20}$$

In order to implement J_F and solve (A.10), J_F can be expressed as the matrix associated with the discretisation of

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}, \beta_1) + a(\beta_1, \mathbf{v}, \mathbf{u}) + d(\mathbf{u}, \mathbf{v}) - b(\mathbf{v}, p) + c(\theta, \mathbf{v}) &= 0 \\ b(\mathbf{u}, q) &= 0 \\ e(\beta_1, \theta, \chi) + f(\theta, \chi) + e(\mathbf{u}, \beta_2, \chi) &= 0 \end{aligned} \tag{A.21}$$

where $\beta_1 = u^{(n)}$, $\beta_2 = \theta^{(n)}$ are known from the previous Newton iteration, indeed J_F is actually evaluated in $X^{(n)}$.

Life Implementation

Now we use the Life non linear framework in order to implement our Newton scheme (A.10). We need to define two new functions in our application

- `updateJacobian(X,J)` which takes as input `X`= $X^{(n)}$ and returns the matrix `J`=$J_F(X^{(n)})$
- `updateResidual(X,R)` which takes as input `X`= $X^{(n)}$ and returns the vector `R`=$F(X^{(n)})$

Remark 5 *Backend Only the PETSC backend supports the nonlinear solver framework. Use in the command line like in the first section*

```
--backend=petsc
```

Here is a snippet of code that implements the nonlinear framework.

```
class MyApp
{
  public:
  void run();
  void updateResidual( const vector_ptrtype& X, vector_ptrtype& R );
  void updateJacobian( const vector_ptrtype& X, sparse_matrix_ptrtype& J);
  void solve( sparse_matrix_ptrtype& D, element_type& u, vector_ptrtype& F );
  private:

  backend_ptrtype M_backend;
  sparse_matrix_ptrtype M_jac;
  vector_ptrtype M_residual;
};

void
MyApp::run()
{
  // ...

  // plug the updateResidual and updateJacobian functions
  // in the nonlinear framework
  M_backend->nlSolver()->residual = boost::bind( &self_type::updateResidual,
                                                 boost::ref( *this ), _1, _2 );
  M_backend->nlSolver()->jacobian = boost::bind( &self_type::updateJacobian,
                                                 boost::ref( *this ), _1, _2 );

  vector_ptrtype U( M_backend->newVector( u.functionSpace() ) );
  *U = u;
  vector_ptrtype R( M_backend->newVector( u.functionSpace() ) );
  this->updateResidual( U, R );
  sparse_matrix_ptrtype J;
  this->updateJacobian( U, J );
  solve( J, u, R );

  *U = u;
  this->updateResidual( U, R );
  // R(u) should be small
  std::cout << "R( u ) = " << M_backend->dot( U, R ) << "\n";


}
void
MyApp::solve( sparse_matrix_ptrtype& D, element_type& u, vector_ptrtype& F )
{
  vector_ptrtype U( M_backend->newVector( u.functionSpace() ) );
  *U = u;
  M_backend->nlSolve( D, U, F, 1e-10, 10 );
  u = *U;
}
void
MyApp::updateResidual( const vector_ptrtype& X, vector_ptrtype& R )
{
  // compute R(X)

  R=M_residual;
}
void
MyApp::updateJacobian( const vector_ptrtype& X, vector_ptrtype& R )
{
  // compute J(X)

  J=M_jac;
}
```

see `bratu.cpp` or `nonlinearpow.cpp` for example.

A.3 Weak Dirichlet boudary conditions

A.3.1 Basic idea

Weak treatment

In order to treat the boundary conditions uniformly (i.e. the same way as Neumann and Robin Conditions), we wish to treat the Dirichlet BC (e.g. $u = g$) weakly.

Remark 6 *Initial Idea add the penalisation term $\int_{\partial\Omega} \mu(u - g)$ where μ is a constant. But this is not enough, this is not consistent with the initial formulation.*

One can use the Nitsche "trick" to implement weak Dirichlet conditions.

- write the equations in conservative form (i.e. identify the flux);
- add the terms to ensure consistency (i.e the flux on the boundary);
- symmetrize to ensure adjoint consistency;
- add a penalisation term with factor $\gamma(u - g)/h$ that ensures that the solution will be set to the proper value at the boundary;

Penalisation parameter

Remark 7 *Choosing γ γ must be chosen such that the coercivity(or inf-sup) property is satisfied. Difficult to do in general. Increase γ until the BC are properly satisfied, e.g. start with $\gamma = 1$, typical values are between 1 and 10.*
The choice of γ is a problem specially when h is small.

Advantages, disadvantages

Remark 8 *Weak treatment: Advantages*

- *uniform(weak) treatment of all boundary conditions type*
- *if boundary condition is independant of time, the terms are assembled once for all*
- *the boundary condition is not enforced exactely but the convergence order remain optimal*

Remark 9 *Weak treatment: Disadvantages*

- *Introduction of the penalisation parameter γ that needs to be tweaked*

Advantages, disadvantages

Remark 10 *Strong treatment: Advantages*

- *Enforce exactely the boundary conditions*

Remark 11 *Strong treatment : Disadvantages*

- *Need to modify the matrix once assembled to reflect that the Dirichlet degree of freedom are actually known. Then even if the boundary condition is independant of time, at every time step if there are terms depending on time that need reassembly (e.g. convection) the strong treatment needs to be reapplied.*
- *it can be expensive to apply depending on the type of sparse matrix used, for example using CSR format setting rows to 0 except on the diagonal to 1 is not expensive but one must do that also for the columns associated with each Dirichlet degree of freedom and that is expensive.*

A.3.2 Laplacian

Example: Laplacian

$$-\Delta u = f(\text{non conservative}),\ -\nabla\cdot(\nabla u) = f(\text{conservative}),\ u = g|_{\partial\Omega} \tag{A.22}$$

the flux is vector ∇u

$$\int_\Omega \nabla u\cdot\nabla v + \int_{\partial\Omega} \underbrace{-\frac{\partial u}{\partial n}v}_{\text{integration by part}} \underbrace{-\frac{\partial v}{\partial n}u}_{\text{adjoint consistency: symetrisation}} + \underbrace{\frac{\gamma}{h}uv}_{\text{penalisation: enforce Dirichlet condition}} \tag{A.23}$$

$$\int_\Omega f\nabla v + \int_{\partial\Omega} (\underbrace{-\frac{\partial v}{\partial n}g}_{\text{adjoint consistency}} + \underbrace{\frac{\gamma}{h}v)g}_{\text{penalisation: enforce Dirichlet condition}} \tag{A.24}$$

Example: Laplacian

```
// bilinear form (left hand side)
form2( Xh, Xh, D ) +=
integrate( boundaryfaces(mesh), im_type(),
           -(gradt(u)*N())*id(v) // integration by part
           -(grad(v)*N())*idt(u) // adjoint consistency
           +gamma*id(v)*idt(u)/hFace()); // penalisation
// linear form (right hand side)
form1( Xh, F ) +=
integrate( boundaryfaces(mesh), im_type(),
           -(grad(v)*N())*g // adjoint consistency
           +gamma*id(v)*g/hFace()); // penalisation
```

A.3.3 Convection-Diffusion

Example: Convection-Diffusion

Remark 12 *Convection Diffusion Consider now the following problem, find u such that*

$$-\Delta u + \mathbf{c}\cdot\nabla u = f,\quad u = g|_{\partial\Omega},\quad \nabla\cdot\mathbf{c} = 0 \tag{A.25}$$

under conservative form the equation reads

$$\nabla\cdot(-\nabla u + \mathbf{c}u) = f,\quad u = g|_{\partial\Omega},\quad \nabla\cdot\mathbf{c} = 0 \tag{A.26}$$

the flux vector field is $\mathbf{F} = -\nabla u + \mathbf{c}u$. Note that here the condition, $\nabla\cdot\mathbf{c} = 0$ was crucial to expand $\nabla\cdot(\mathbf{c}u)$ into $\mathbf{c}\cdot\nabla u$ since

$$\nabla\cdot(\mathbf{c}u) = \mathbf{c}\cdot\nabla u + \underbrace{u\nabla\cdot\mathbf{c}}_{=0} \tag{A.27}$$

Weak formulation for convection diffusion

Multiplying by any test function v and integration by part of (A.26) gives

$$\int_\Omega \nabla u\cdot\nabla v + (\mathbf{c}\cdot\nabla u)v + \int_{\partial\Omega}(\mathbf{F}\cdot\mathbf{n})v = \int_\Omega fv \tag{A.28}$$

where $\mathbf{n}$ is the outward unit normal to $\partial\Omega$. We now introduce the penalisation term that will ensure that $u\to g$ as $h\to 0$ on $\partial\Omega$. (A.28) reads now

$$\int_\Omega \nabla u\cdot\nabla v + (\mathbf{c}\cdot\nabla u)v + \int_{\partial\Omega}(\mathbf{F}\cdot\mathbf{n})v + \frac{\gamma}{\mathbf{h}}\mathbf{uv} = \int_\Omega fv + \int_{\partial\mathbf{\Omega}}\frac{\gamma}{\mathbf{h}}\mathbf{gv} \tag{A.29}$$

Finally we incorporate the symetrisation of the bilinear form to ensure adjoint consistency and hence proper convergence order

$$\int_\Omega \nabla u \cdot \nabla v + (\mathbf{c} \cdot \nabla u)v + \int_{\partial\Omega} ((-\nabla u + \mathbf{c}u) \cdot \mathbf{n})v + ((-\nabla \mathbf{v} + \mathbf{c}\mathbf{v}) \cdot \mathbf{n})\mathbf{u} + \frac{\gamma}{h}uv = \int_\Omega fv + \int_{\partial\Omega} ((-\nabla \mathbf{v} + \mathbf{c}\mathbf{v}) \cdot \mathbf{n})\mathbf{g} + \frac{\gamma}{h}gv \tag{A.30}$$

Example: Convection-Diffusion

```
// bilinear form (left hand side)
form2( Xh, Xh, D ) +=
integrate( boundaryfaces(mesh), im_type(),
           // integration by part
           -(gradt(u)*N())*id(v) + (idt(u)*trans(idv(c))*N())*id(v)
           // adjoint consistency
           -(grad(v)*N())*idt(u) + (id(v)*trans(idv(c))*N())*idt(u)
           // penalisation
           +gamma*id(v)*idt(u)/hFace());
// linear form (right hand side)
form1( Xh, F ) +=
integrate( boundaryfaces(mesh), im_type(),
           // adjoint consistency
           -(grad(v)*N())*g + (id(v)*trans(idv(c))*N())*g
           // penalisation
           +gamma*id(v)*g/hFace());
```

A.3.4 Stokes

Example: Stokes

Remark 13 *Stokes Consider now the following problem, find* $(\mathbf{u}, p)$ *such that*

$$-\Delta\mathbf{u} + \nabla p = \mathbf{f}, \quad \mathbf{u} = \mathbf{g}|_{\partial\Omega}, \quad \nabla \cdot \mathbf{u} = 0 \tag{A.31}$$

under conservative form the equation reads

$$\nabla \cdot (-\nabla\mathbf{u} + p\mathbb{I}) = \mathbf{f}, \tag{A.32}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{A.33}$$

$$\mathbf{u} = \mathbf{g}|_{\partial\Omega} \tag{A.34}$$

where $\mathbb{I}(\mathbf{x}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ *(in 2D)* $\forall \mathbf{x} \in \Omega$ *is the identity tensor(matrix) field* $\in \mathbb{R}^{d\times d}$*. The flux tensor field is* $\mathbf{F} = -\nabla\mathbf{u} + p\mathbb{I}$*. Indeed we have the following relation, if* $\mathbb{M}$ *is a tensor (rank 2) field and* $\mathbf{v}$ *is a vector field*

$$\nabla \cdot (\mathbb{M}\mathbf{v}) = (\nabla \cdot \mathbb{M}) \cdot \mathbf{v} + \mathbb{M} : (\nabla\mathbf{v}) \tag{A.35}$$

where $\mathbb{M} : (\nabla\mathbf{v}) = \mathrm{trace}(\mathbb{M} * \nabla\mathbf{v}^T)$*,* $*$ *is the matrix-matrix multiplication and* $\nabla \cdot \mathbb{M}$ *is the vector field with components the divergence of each row of* $\mathbb{M}$*. For example* $\nabla \cdot (p\,\mathbb{I}) = \nabla \cdot \begin{pmatrix} p & 0 \\ 0 & p \end{pmatrix}$ *(in 2D)* $= \nabla p$*.*

Weak formulation for Stokes

Taking the scalar product of (A.32) by any test function $\mathbf{v}$ (associated to velocity) and multiplying (A.33) by any test function q (associated to pressure), the variational formulation of (A.32) reads, thanks to (A.35),

$$\int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v} + p\nabla \cdot \mathbf{v} + \int_{\partial\Omega} ((-\nabla\mathbf{u} + p\mathbb{I})\mathbf{n}) \cdot \mathbf{v} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \tag{A.36}$$

where $\mathbf{n}$ is the outward unit normal to $\partial\Omega$. We now introduce the penalisation term that will ensure that $\mathbf{u} \to \mathbf{g}$ as $h \to 0$ on $\partial\Omega$. (A.36) reads now

$$\int_\Omega \nabla\mathbf{u} \colon \nabla\mathbf{v} + p\nabla\cdot\mathbf{v} + \int_{\partial\Omega} ((-\nabla\mathbf{u} + p\mathbb{I})\mathbf{n})\cdot\mathbf{v} + \frac{\gamma}{\mathbf{h}}\mathbf{u}\cdot\mathbf{v} = \int_\Omega \mathbf{f}\cdot\mathbf{v} + \int_{\partial\mathbf{\Omega}} \frac{\gamma}{\mathbf{h}}\mathbf{g}\cdot\mathbf{v} \tag{A.37}$$

Finally we incorporate the symetrisation of the bilinear form to ensure adjoint consistency and hence proper convergence order

$$\begin{aligned}\int_\Omega \nabla\mathbf{u} \colon \nabla\mathbf{v} + p\nabla\cdot\mathbf{v} + \int_{\partial\Omega} ((-\nabla\mathbf{u} + p\mathbb{I})\mathbf{n})\cdot\mathbf{v} + ((-\nabla\mathbf{v} + q\mathbb{I})\mathbf{n})\cdot\mathbf{u} + \frac{\gamma}{h}\mathbf{u}\cdot\mathbf{v} = \\ \int_\Omega \mathbf{f}\cdot\mathbf{v} + \int_{\partial\Omega} ((-\nabla\mathbf{v} + q\mathbb{I})\mathbf{n})\cdot\mathbf{g} + \frac{\gamma}{h}\mathbf{g}\cdot\mathbf{v}\end{aligned} \tag{A.38}$$

Example: Stokes

```
// total stress tensor (trial)
AUTO( SigmaNt, (-idt(p)*N()+mu*gradt(u)*N()) );
// total stress tensor (test)
AUTO( SigmaN, (-id(p)*N()+mu*grad(v)*N()) );
// linear form (right hand side)
form1( Xh, F ) +=
integrate( boundaryfaces(mesh), im,
           trans(g)*(-SigmaN+gamma*id(v)/hFace() ) );
// bilinear form (left hand side)
form2( Xh, Xh, D )+=
integrate( boundaryfaces(mesh), im,
           -trans(SigmaNt)*id(v)
           -trans(SigmaN)*idt(u)
           +gamma*trans(idt(u))*id(v)/hFace() );
```

A.4 Stabilisation techniques

A.4.1 Convection dominated flows

Consider this type of problem

$$-\epsilon\Delta u + \mathbf{c}\cdot\nabla u + \gamma u = f, \quad \nabla\cdot\mathbf{c} = 0 \tag{A.39}$$

Introduce $\mathrm{Pe} = \frac{|\mathbf{c}|h}{\epsilon}$ the *Péclet* number. The dominating convection occurs when, on at least some cells, $\mathrm{Pe} >> 1$. We talk about singularly (i.e. $\epsilon << h$) perturbed flows.

Without doing anything wiggles occur. There are remedies so called *Stabilisation Methods*, here some some examples:

- Artificial diffusion (streamline diffusion) (SDFEM)
- Galerkin Least Squares method (GaLS)
- Streamline Upwind Petrov Galerkin (SUPG)
- Continuous Interior Penalty methods (CIP)

A.4.2 The CIP methods

Add the term

$$\sum_{F\in\Gamma_{\mathrm{int}}} \int_F \gamma\, h_F^2\, |\mathbf{c}\cdot\mathbf{n}|\, [\nabla u][\nabla v] \tag{A.40}$$

where Γ_{int} is the set of internal faces where the $\text{Pe} >> 1$ (typically it is applied to all internal faces) and

$$[\nabla u] = \nabla u \cdot \mathbf{n}|_1 + \nabla u \cdot \mathbf{n}|_2 \tag{A.41}$$

is the jump of ∇u(scalar valued) across the face. In the case of scalar valued functions

$$[u] = u\mathbf{n}|_1 + u\mathbf{n}|_2 \tag{A.42}$$

Remark 14 (Choice for γ) *γ can be taken in the range $[1e-2; 1e-1]$. A typical value is $2.5e-2$.*

```
    // define the stabilisation coefficient expression
    AUTO( stab_coeff , (γβ abs(trans(N())*idv(beta)))*
                        vf::pow(hFace(),2.0));

    // assemble the stabilisation operator
    form2( Xh, Xh, M ) +=
     integrate(
        // internal faces of the mesh
        internalfaces(Xh->mesh()),
        // integration method
        _Q<OrderOfPolynomialToBeIntegratedExactely>,
        // stabilisation term
        stab_coeff*(trans(jumpt(gradt(u)))*jump(grad(v))));
```

A.5 Interpolation

In order to interpolate a function defined on one domain to another domain, one can use the `interpolate` function. The basis function of the image space must be of `Lagrange` type.

```
typedef bases<Lagrange<Order, Vectorial> > basis_type; // velocity
typedef FunctionSpace<mesh_type, basis_type, value_type> space_type;
// ...
space_ptrtype Xh = space_type::New( mesh1 );
element_type u( Xh, "u" );
space_ptrtype Yh = space_type::New( mesh2 );
element_type v( Yh, "v" );

// interpolate u on mesh2 and store the result in v
interpolate( Yh, u, v );
```

Appendix B

FEEL++

One of Life assets is it finite element embedded language (FEEL++). The language follows the C++ grammar, and provides keywords as well as operations between objects which are, mathematically, tensors of rank 0, 1 or 2.

B.1 Predefined functions

Some notations

- $f : \mathbb{R}^n \mapsto \mathbb{R}^{m \times p}$ with $n = 1, 2, 3$, $m = 1, 2, 3$, $p = 1, 2, 3$.
- Ω^e current mesh element

Keyword	Math object	Description	Rank	$M \times N$
`P()`	$\overrightarrow{P}$	current point coordinates $(P_x, P_y, P_z)^T$	1	$d \times 1$
`Px()`	P_x	x coordinate of $\overrightarrow{P}$	0	1×1
`Py()`	P_y	y coordinate of $\overrightarrow{P}$ (value is 0 in 1D)	0	1×1
`Pz()`	P_z	z coordinate of $\overrightarrow{P}$ (value is 0 in 1D and 2D)	0	1×1
`C()`	$\overrightarrow{C}$	element barycenter point coordinates $(C_x, C_y, C_z)^T$	1	$d \times 1$
`Cx()`	C_x	x coordinate of $\overrightarrow{C}$	0	1×1
`Cy()`	C_y	y coordinate of $\overrightarrow{C}$ (value is 0 in 1D)	0	1×1
`Cz()`	C_z	z coordinate of $\overrightarrow{C}$ (value is 0 in 1D and 2D)	0	1×1
`N()`	$\overrightarrow{N}$	normal at current point $(N_x, N_y, N_z)^T$	1	$d \times 1$
`Nx()`	N_x	x coordinate of $\overrightarrow{N}$ at current point	0	1×1
`Ny()`	N_y	y coordinate of $\overrightarrow{N}$ at current point (value is 0 in 1D)	0	1×1
`Nz()`	N_z	z coordinate of $\overrightarrow{N}$ at current point (value is 0 in 1D and 2D)	0	1×1
`eid()`	e	index of Ω^e	0	1×1

Keyword	Math object	Description	Rank	$M \times N$
`emarker()`	$m(e)$	marker of Ω^e	0	1×1
`h()`	h^e	size of Ω^e	0	1×1
`hFace()`	h^e_Γ	size of face Γ of Ω^e	0	1×1
`mat<M,N>(m_11, m_12,...)`	$\begin{pmatrix} m_{11} & m_{12} & \dots \\ m_{21} & m_{22} & \dots \\ \vdots & & \end{pmatrix}$	$M \times N$ matrix entries being expressions	2	$M \times N$
`vec<M>(v_1, v_2,...)`	$(v_1, v_2, ...)^T$	column vector with M rows entries being expressions	1	$M \times 1$
`trace(expr)`	$\mathrm{tr}(f(\overrightarrow{x}))$	trace of $f(\overrightarrow{x})$	0	1×1
`abs(expr)`	$\lvert f(\overrightarrow{x})\rvert$	element wise absolute value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`cos(expr)`	$\cos(f(\overrightarrow{x}))$	element wise cosinus value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`sin(expr)`	$\sin(f(\overrightarrow{x}))$	element wise sinus value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`tan(expr)`	$\tan(f(\overrightarrow{x}))$	element wise tangent value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`acos(expr)`	$\mathrm{acos}(f(\overrightarrow{x}))$	element wise acos value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`asin(expr)`	$\mathrm{asin}(f(\overrightarrow{x}))$	element wise asin value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`atan(expr)`	$\mathrm{atan}(f(\overrightarrow{x}))$	element wise atan value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`cosh(expr)`	$\cosh(f(\overrightarrow{x}))$	element wise cosh value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`sinh(expr)`	$\sinh(f(\overrightarrow{x}))$	element wise sinh value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`tanh(expr)`	$\tanh(f(\overrightarrow{x}))$	element wise tanh value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`exp(expr)`	$\exp(f(\overrightarrow{x}))$	element wise exp value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`log(expr)`	$\log(f(\overrightarrow{x}))$	element wise log value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`sqrt(expr)`	$\sqrt{f(\overrightarrow{x})}$	element wise sqrt value of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`sign(expr)`	$\begin{cases} 1 & \text{if } f(\overrightarrow{x}) \geq 0 \\ -1 & \text{if } f(\overrightarrow{x}) < 0 \end{cases}$	element wise sign of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`chi(expr)`	$\chi(f(\overrightarrow{x})) = \begin{cases} 0 & \text{if } f(\overrightarrow{x}) = 0 \\ 1 & \text{if } f(\overrightarrow{x}) \neq 0 \end{cases}$	element wise boolean test of f	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`id(f)`	f	test function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`idt(f)`	f	trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`idv(f)`	f	evaluation function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
`grad(f)`	∇f	gradient of test function	$\mathrm{rank}(f(\overrightarrow{x})) + 1$	$p = 1,\ m \times$
`gradt(f)`	∇f	gradient of trial function	$\mathrm{rank}(f(\overrightarrow{x})) + 1$	$p = 1,\ m \times$
`gradv(f)`	∇f	evaluation function gradient	$\mathrm{rank}(f(\overrightarrow{x})) + 1$	$p = 1,\ m \times$
`div(f)`	$\nabla \cdot \overrightarrow{f}$	divergence of test function	$\mathrm{rank}(f(\overrightarrow{x})) - 1$	1×1[2]
`divt(f)`	$\nabla \cdot \overrightarrow{f}$	divergence of trial function	$\mathrm{rank}(f(\overrightarrow{x})) - 1$	1×1
`divv(f)`	$\nabla \cdot \overrightarrow{f}$	evaluation of function divergence	$\mathrm{rank}(f(\overrightarrow{x})) - 1$	1×1
`curl(f)`	$\nabla \times \overrightarrow{f}$	curl of test function	1	$n = m, n \times$
`curlt(f)`	$\nabla \times \overrightarrow{f}$	curl of trial function	1	$m = n, n \times$
`curlv(f)`	$\nabla \times \overrightarrow{f}$	evaluation of function curl	1	$m = n, n \times$
`hess(f)`	$\nabla^2 f$	hessian of test function	2	$m = p = 1,$
`jump(f)`	$[f] = f_0\overrightarrow{N_0} + f_1\overrightarrow{N_1}$	jump of test function	1	$m = 1, n \times$
`jump(f)`	$[\overrightarrow{f}] = \overrightarrow{f_0} \cdot \overrightarrow{N_0} + \overrightarrow{f_1} \cdot \overrightarrow{N_1}$	jump of test function	0	$m = 2, 1 \times$

[1] Gradient of matrix value functions is not implemented, hence $p = 1$

[2] Divergence of matrix value functions is not implemented, hence $p = 1$

Keyword	Math object	Description	Rank	$M \times N$
jumpt(f)	$[f] = f_0\overrightarrow{N_0} + f_1\overrightarrow{N_1}$	jump of trial function	1	$m = 1, n \times$
jumpt(f)	$[\overrightarrow{f}] = \overrightarrow{f_0} \cdot \overrightarrow{N_0} + \overrightarrow{f_1} \cdot \overrightarrow{N_1}$	jump of trial function	0	$m = 2, 1 \times$
jumpv(f)	$[f] = f_0\overrightarrow{N_0} + f_1\overrightarrow{N_1}$	jump of function evaluation	1	$m = 1, n \times$
jumpv(f)	$[\overrightarrow{f}] = \overrightarrow{f_0} \cdot \overrightarrow{N_0} + \overrightarrow{f_1} \cdot \overrightarrow{N_1}$	jump of function evaluation	0	$m = 2, 1 \times$
average(f)	$f = \frac{1}{2}(f_0 + f_1)$	average of test function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
averaget(f)	$f = \frac{1}{2}(f_0 + f_1)$	average of trial function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
averagev(f)	$f = \frac{1}{2}(f_0 + f_1)$	average of function evaluation	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftface(f)	f_0	left test function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftfacet(f)	f_0	left trial function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftfacev(f)	f_0	left function evaluation	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightface(f)	f_1	right test function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightfacet(f)	f_1	right trial function	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightfacev(f)	f_1	right function evaluation	$\text{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
maxface(f)	$\max(f_0, f_1)$	maximum of right and left test function	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
maxfacet(f)	$\max(f_0, f_1)$	maximum of right and left trial function	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
maxfacev(f)	$\max(f_0, f_1)$	maximum of right and left function evaluation	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
minface(f)	$\min(f_0, f_1)$	minimum of right and left test function	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
minfacet(f)	$\min(f_0, f_1)$	minimum of right and left trial function	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
minfacev(f)	$\min(f_0, f_1)$	minimum of right and left function evaluation	$\text{rank}(f(\overrightarrow{x}))$	$m \times p$
-	$-g$	element wise unary minus		
-	$!g$	element wise logical not		
+	$f + g$	tensor sum		
-	$f - g$	tensor substraction		
*	$f * g$	tensor product		
/	f/g	tensor division (g scalar field)		
<	$f < g$	element wise less		
<=	$f \leq g$	element wise less or equal		
>	$f > g$	element wise greater		
>=	$f \geq g$	element wise greater or equal		
==	$f = g$	element wise equal		
!=	$f \neq g$	element wise not equal		
&&	f and g	element wise logical and		
||	f or g	element wise logical or		

Appendix C

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

Verbatim copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover,

and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

3. State on the Title page the name of the publisher of the Modified Version, as the publisher.

4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

8. Include an unaltered copy of this License.

9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

Collections of documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation with independent works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

Future revisions of this license

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been

published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

Bibliography

[1] Wikipedia. http://fr.wikipedia.org.

[2] J.-L. GUERMOND. *Fluid Mechanics: Numerical Methods.*, pages 365–374. Oxford: Elsevier, Ecyclopedia of Mathematical Physics, eds. J.-P. Françoise, G.L. Naber, and Tsou S.T. , (ISBN 978-0-1251-2666-3), volume 2 edition, 2006.

[3] Incropera, DeWitt, Bergman, and Lavine. *Fundamentals of heat and mass transfer*. Wiley, 6 edition, 2007.

[4] Christophe Prud'homme. A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations. *Scientific Programming*, 14(2):81–110, 2006. http://iospress.metapress.com/link.asp?id=8xwd8r59hg1hmlcl.

[5] Christophe Prud'homme. Life: Overview of a unified c++ implementation of the finite and spectral element methods in 1d, 2d and 3d. In *Workshop On State-Of-The-Art In Scientific And Parallel Computing*, Lecture Notes in Computer Science, page 10. Springer-Verlag, 2006. Accepted.