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

Version 0.92.0 gitbe2c5ee

Editor
Christophe PRUD'HOMME
FEEL++ Consortium
christophe.prudhomme@feelpp.org

Contents

Part I

Tutorial

Vincent Huber

<div style="text-align: right">

CHAPTER **1**

</div>

<div style="text-align: right">

Crash course

By Vincent Huber

</div>

Chapter ref: **[cha:tutorial-crash-course]**

This chapter is designed for impatient people who wants to test FEEL++ as soon as possible.

1.1 Requirements

Before installing FEEL++, you need to get this required packages:

- g++ (4.4, 4.5, 4.6 and 4.7)
- MPI : openmpi (preferred) or mpich
- Boost (≥ 1.39)
- Petsc ($\geq 2.3.3$)
- Cmake (≥ 2.6)
- Gmsh[1]
- Libxml2

1.2 Building FEEL++ from source on *nix

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

<div style="text-align: center">

http://code.google.com/p/feelpp/downloads/list

</div>

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
```

We define then the current directory as the source one, ie:

```
export FeelppSrcDir=`pwd`
```

---

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

1.3 Compiling

In order to compile FEEL++ and a test application, we create a new directory:

```
cd $FeelppSrcDir
cd ..
mkdir FeelBin
cd FeelBin
export FeelBinDir=`pwd`
```

and then, we are able to compile our first application:

```
cd $FeelppBinDir
cmake $FeelppSrcDir
cd quickstart
make feelpp_qs_laplacian
```

It is recommended to compile on multiple threads:

```
make -j8 feelpp_qs_laplacian
```

1.4 FEEL++ Hello World

As an introduction to the aim and the way to do with FEEL++, we provide a sort of *Hello World* program to evaluate the library.

1.4.1 About the math

We want to solve the simplest problem:

$$-\Delta u = 1,$$
$$u_{|\partial\Omega} = 0,$$

where $\Omega \in \mathbb{R}^n, n \in 1, 2, 3$.

That problem is written, in the Finite Element Method formulation:

$$a\left(u, v\right) = f(v).$$

with:

$$a\left(u, v\right) = \int_{\Omega} \nabla u \cdot \nabla v dx,$$
$$f\left(v\right) = \int_{\Omega} v dx.$$

The aim of FEEL++ is to provide the simplest way to write the a and f forms.

1.4.2 About the code

This section is here to declare that we want to use the namespace FEEL++, to passe the command line options to the created environnement and add some informations (basics FEEL++ options, application name).

```
using namespace Feel;
    Environment env( _argc=argc, _argv=argv,
                    _desc=feel_options(),
                    _about=about(_name="qs_laplacian",
                                 _author="Feel++ Consortium",
                                 _email="feelpp-devel@feelpp.org"));
```

We have to define the mesh, the approximation space and our test and trial functions.

```cpp
auto mesh = unitSquare();
auto Vh = Pch<1>( mesh );
auto u = Vh->element();
auto v = Vh->element();
```

We create now our bilinear and linear forms, we add the homogeneous Dirichlet conditions and solve the discretized (linear) system.

```cpp
auto l = form1( _test=Vh );
l = integrate(_range=elements(mesh),
              _expr=id(v));

auto a = form2( _trial=Vh, _test=Vh );
a = integrate(_range=elements(mesh),
              _expr=gradt(u)*trans(grad(v)) );
a+=on(_range=boundaryfaces(mesh), _rhs=l, _element=u,
      _expr=constant(0.) );
a.solve(_rhs=l,_solution=u);
```

FEEL++ provides the possibility to save the results:

```cpp
auto e = exporter( _mesh=mesh );
e->add( "u", u );
e->save();
return 0;
```

1.5   First execution & vizualisation

To test that part of code, please go to:

$Feelpp_src/quickstart

and execute the code, by:

./feelpp_qs_laplacian

This will produce several files:

qs_laplacian-1_0.case
qs_laplacian-1.sos
qs_laplacian.timeset
qs_laplacian.u-1_0.001
qs_laplacian-1_0.geo001
qs_laplacian.pid-1_0.001
square.geo
square.msh

You can vizualise the results using any Ensight file reader, such as Paraview, opening `qs_laplacian-1.sos`.

paraview qs_laplacian-1.sos

CHAPTER 2

Building Feel++

By Christophe Prud'homme, Baptiste Morin

Chapter ref: **[cha:tutorial-building]**

2.1 Building FEEL++

2.1.1 Getting the source via an archive

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

<center>http://www.feelpp.org/files</center>

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
```

2.1.2 Getting the source via Subversion

In order to download the sources of FEEL++, you can download it directly from the source depository thanks to Subversion. To make it possible, you can download them anonymously or with an account in LJKForge that you have created. As an open-source project, we strongly suggest you to create an account and take part of the project with sharing your ideas, developments or suggests. If you're interested to participate and become a FEEL++ developer, please don't hesitate to see how it works in the appendix B.1. For now, if you want to get the sources without an account, open a command-line and type

```
svn checkout svn://scm.forge.imag.fr/var/lib/gforge/chroot/scmrepos/svn/life/trunk/life/trunk feel
```

then you can go to the FEEL++ top directory with

```
cd feel
```

You should obtain furthers directories such as :

```
applications/   # functional applications
benchmarks/  # applications under test
cmake/   # do not touch, used for compilation
contrib/
doc/   # tutorial and examples
```

```
examples/ # examples using Feel++
feel/   # Feel++ library
ports/   # used for Mac OS X installation
research/   # research projects using Feel++
testsuite/ # Feel++ unit tests testsuite
CMakeListe.txt   # the file for cmake to build, do not modify
...
```

2.1.3 Unix : dependencies

In order to install FEEL++ on Unix systems (other than Mac OS X, in you have a Macintosh, please go to 2.1.5), 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.
Required packages:

- g++ (4.4, 4.5, 4.6 and 4.7)

- MPI : openmpi (preferred) or mpich

- Boost ($\geq$1.39)

- Petsc ($\geq$2.3.3)

- Cmake ($\geq$2.6)

- Gmsh[1]

- Libxml2

Optional packages:

- Superlu

- Suitesparse(umfpack)

- Metis: scoth with the metis interface (preferred), metis (non-free)

- Trilinos ($\geq$8.0.8)

- Google perftools

- Paraview[2], this is not stricly required to run FEEL++ programs but it is somehow necessary for visualisation

- Python ($\geq$ 2.5) for the validation tools

Note that all these packages are available under Debian/GNU/Linux and Ubuntu. They should be available. Once you have installed those dependencies, you can jump to 2.1.6.

2.1.4 FEEL++ on Debian and Ubuntu

Debian

Debian is the platform of choice for FEEL++, it was developed mainly on it. The commands to install FEEL++ on Debian are

```
sudo apt-get update
sudo apt-get install feel++-apps libfeel-dev feel++-doc
```

---

[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

The interested user is encourage to follow the FEEL++ PTS page

- FEEL++ Debian Packages Tracking System

At the moment FEEL++ compiles and is available on the following Debian plateforms:

- FEEL++ Buildd results

Ubuntu

FEEL++ was uploaded in the distribution Ubuntu-Natty (11.04) for the first time. The commands to install FEEL++ on Ubuntu are

```
sudo apt-get update
sudo apt-get install feel++-apps libfeel-dev feel++-doc
```

The interested user might want to follow the Ubuntu Launchpad FEEL++ page in order to know what is going on with FEEL++ on Ubuntu

- FEEL++ Ubuntu Source Page for all Ubuntu versions

2.1.5 FEEL++ *on Mac OS X*

FEEL++ is also working on Mac operating systems. The way to make it work is quite different.

Compilers

In order to FEEL++ and `cmake` work properly, you have to install differents compilers :

- Gcc
  The first step is to install the latest version of Xcode. If your computer is recent, you can install it with your DVD that came with your machine (not the OS DVD, but the applications one). You don't have to install the complete Xcode (you can uncheck iOS SDK for example, it's not necessary here and requiers a lot of memory). Xcode will provide your computer all basic tools to compile such as gcc 4.2. It's the first step, you'll see later how to easily install gcc 4.5 or later using MacPorts.

- Fortran
  To build the Makefiles, `cmake` will need a Fortran compiler. To make it works, please go to Source-Forge.net and download `gfortran-snwleo-intel-bin.tar.gz` which is the fortran compiler only (from now, don't download the complete install with gcc 4.6 because Feel needs gcc 4.5 or later). To install it, go to the directory where you have downloaded the file and type in a command-line

  ```
  sudo tar -xvf gfortran-snwleo-intel-bin.tar -C /
  ```

MacPorts

Introduction   MacPorts is an open-source community projet which aims to design an easy-to-use system for compiling, installing and upgrading open-source softwares on Mac OS X operating system. It is distributed under BSD License and facilitate the access to thousands of ports (softwares) without installing or compiling open-source softwares. MacPorts provides a single software tree which includes the latest stable releases of approximately 8050 ports targeting the current Mac OS X release (10.6 or 10.5). If you want more information, please visite their website.

Installation To install the latest version of MacPorts, please go to Installing MacPorts page and follow the instructions. The simplest way is to download the *dmg* disk image corresponding to your version of Mac OS X. It is recommended that you install X11 (X Window System) which is normally used to display X11 applications.

If you have installed with the package installer (`MacPorts-1.x.x.dmg`) that means MacPorts will be installed in `/opt/local`. From now on we will suppose that macports has been installed in `/opt/local` which is the default MacPorts location. Note that from now on, all tools installed by MacPorts will be installed in `/opt/local/bin` or `/opt/local/sbin` for example (that's here you'll find gcc4.5 or later e.g `/opt/local/bin/g++-mp-4.5` once being installed).

Key commands In your command-line, the software MacPorts is called by the command `port`. Here is a list of key commands for using MacPorts, if you want more informations please go to MacPorts Commands.

- `sudo port -v selfupdate` This action should be used regularly to update the local tree with the global MacPorts ports. The option `-v` enables verbose which generates verbose messages.

- `port info flowd` This action is used to get information about a port (description, license, maintainer, etc.)

- `sudo port install mypackage` This action install the port mypackage

- `sudo port uninstall mypackage` This action uninstall the port mypackage

- `port installed` This action displays all ports installed and their versions, variants and activation status. You can also use the `-v` option to also display the platform and CPU architecture(s) for which the ports were built, and any variants which were explicitly negated.

- `sudo port upgrade mypackage` This action updgrades installed ports and their dependencies when a `Portfile` in the repository has been updated. To avoid the upgrade of a port's dependencies, use the option `-n`.

Portfile A Portfile is a TCL script which usually contains simple keyword values and TCL expressions. Each package/port has a corresponding Portfile but it's only a part of a port description. FEEL++ provides some mandatory Portfiles for its compilation which are either not available in MacPorts or are buggy but FEEL++ also provides some Portfiles which are already available in MacPorts such as gmsh or petsc. They usually provide either some fixes to ensure FEEL++ works properly or new version not yet available in MacPorts. These Portfiles are installed in `ports/macosx/macports`.

MacPorts and FEEL++

To be able to install FEEL++, add the following line in `/opt/local/etc/macports/source.conf` at the top of the file before any other sources :

```
file:///<path to feel top directory>/ports/macosx/macports
```

Once it's done, type in a command-line :

```
cd <your path to feel top directory>/ports/macosx/macports
portindex -f
```

You should have an output like this :

```
Reading port index in <your path to feel top directory>/ports/macosx/macports
Adding port science/feel++
Adding port science/gmsh
Adding port science/petsc

Total number of ports parsed: 3
Ports successfully parsed: 3
Ports failed: 0
Up-to-date ports skipped: 0
```

Your are now able to type

> sudo port install feel++

It might take some time (possibly an entire day) to compile all the requirements for FEEL++ to compile properly. If you have several cores on your MacBook Pro, iMac or MacBook we suggest that you configure macports to use all or some of them. To do that uncomment the following line in the file `/opt/local/etc/macports/macports.conf`

```
buildmakejobs    0 # all the cores
```

At the end of the `sudo port install feel++`, you have all dependencies installed. To build all the Makefile, `cmake` is automatically launched but can have some libraries may not be found but they are not mandatory for build Feel++, only the features related to the missing libraries will be missing.

PETSc and SLEPc on Snow Leopard and Lion

We have heard about issues with petsc and slepc with some new MacBook Pro with Snow Leopard while they are being installed with the command

> sudo port install feel++

If it's the case, that probably means there is an issue with atlas. If atlas is already installed, you have to unsinstall it (be careful with dependencies, they also have to be uninstalled). Once it's done, you should do

> cd <path to feel top directory>/ports/macosx/macports
> portindex -f

then type in the exact same order :

> sudo port uninstall slepc
> sudo port uninstall petsc
> sudo port install -d petsc
> sudo port install slepc

Then add to you shell script environment (e.g. for Bash shells `.bashrc` or `.profile` or for CSh shells `.tcshrc`)

> # Sh based shell
> export PETSC_DIR=/opt/local/lib/petsc
> export SLEPC_DIR=/opt/local/lib/petsc
>
> # CSh based shell
> setenv PETSC_DIR /opt/local/lib/petsc
> setenv SLEPC_DIR /opt/local/lib/petsc

and type once again

> sudo port install feel++

In that order, slepc and petsc will be installed before atlas, and feel will be properly installed.

Missing ports

cmake can build Makefiles even if some packages are missing (latex2html, VTK ...). It's not necessary to install them but you can complete the installation with MacPorts, cmake will find them by itself once they have been installed.

2.1.6 Compiling Feel

Feel build system uses cmake [3] as its build system. Check that cmake is using gcc4.5 or 4.4 as C++ compiler (you can use the option CMAKE_CXX_COMPILER=<path>/g++-4.5 where the path depends on your OS, it's probably /usr/bin or /opt/local/bin but you can also change it with the command ccmake and press t for advanced options). It's important, as cmake did not produce any Makefile, a CMakeCache.txt won't be created so you'll have to check each time that gcc 4.5 is the C++ compiler to be sure the build will be correct.

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)

CMake Out Source Build (preferred) The best way is to have a directory (FEEL for example) in which you have :

```
    feel/
```

where feel is the top directory where the source have been downloaded. Placed in FEEL, you can create the build directory (feel.opt for example) and lauch cmake with :

```
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
```

you can customize the build type:

```
# Choose g++ release
cmake -CMAKE_CXX_COMPILER=/usr/bin/g++-4.5
# Debug build type (-g...)
cmake -D CMAKE_BUILD_TYPE=Debug
# Release build type (-O3...)
cmake -D CMAKE_BUILD_TYPE=Release
...
```

Once Cmake has made its work, you are now able to compile the library with

```
make
```

Important : from now, all commands should be type in feel.opt or its subdirectories.

---

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

CMake In Source Build   Be carefull, this is not advised and if you try this way, `cmake` won't let you do. If you really want to, you will have to modifiy the top `CMakeLists.txt`. You should consider out source builds by checking the 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
```

Compiling the Feel tutorial

The manual (which includes the tutorial) is edited with LATEXso you need to have installed the LATEXdistribution on your computer. LATEXis a high-quality typesetting system, it includes features designed for the production of technical and scientific documentation. There are several ways to make it work, for example you can go on MacTeX website and follow the instructions to install the distribution. If the command `make check` in `feel.opt/` has been run before, the tutorial should be already compiled and ready. The steps are as follows to build the Feel tutorial

```
cd feel.opt/doc/manual
make pdf
```

The directory `doc/manual` contains all examples used in the tutorial. You will see how it works in the following parts.

2.2   Programming environment

We present here a quick list of all namespaces and librairies proposed by FEEL++, you'll see in the tutorial which starts at section **??** how you can use them.

2.2.1   Boost C++ Libraries

FEEL++ depends on a number of libraries, some are required some are optional. Among the required libraries, The Boost C++ libraries play a very important role as they drive or shape the design of FEEL++. FEEL++ uses in particular the following Boost libraries:

- Boost.Parameter : use to provide powerful interfaces to FEEL++ and third party library such as

  - PETSc for the linear, nonlinear solvers
  - SLEPc for the eigenvalue solvers
  - GMSH for mesh generation

- Boost.MPL - meta programming library : use for type computations

- Boost.Fusion - linking meta-runtime programming: use for type computations used at runtime

- Boost.Program_Options - command-line options library : provides the command line options for the FEEL++ applications as well as configuration files

- Boost.Test - Unit testing framework ; used by the FEEL++ testsuite

2.2.2 FEEL++ *Namepaces*

- `Feel`

- `Feel::po`

- `Feel::mpl`

- `Feel::ublas`

- `Feel::math`

- `Feel::fem`

- `Feel::vf`

Christophe Prud'homme, Baptiste Morin, Guillaume Dollé

CHAPTER 3

Getting Started with Feel++

By Christophe Prud'homme, Baptiste Morin, Guillaume Dollé

Chapter ref: **[cha:getting-started]**

3.1 First FEEL++ Application

See section **??** for more information about FEEL++ installation.

3.1.1 Minimal example

Let's begin with our first program using the FEEL++ framework (source "doc/manual/tutorial/myapp.cpp").
Before all, you have to include the FEEL++ headers.

```
#include <feel/feel.hpp>
using namespace Feel;
```

We use the C++ **namespace** to avoid `Feel::` prefix before FEEL++ objects.

```cpp
int main( int argc, char* argv[] )
{
    // create custom command option
    po::options_description app_options( "MyApp options" );
    app_options.add( feel_options() );
    app_options.add_options()
        ( "value",
      po::value<double>() -> default_value(4.2),
      "a 'double' with default value" )
    ;

    // initialize feel++ environment
    Environment env( _argc=argc, _argv=argv,
                     _desc=app_options,
                     _about=about( _name="myapp",
                                   _author="Feel++ Consortium",
                                   _email="feelpp-devel@feelpp.org") );

    // create a log and write inside
    LOG(INFO) << "value = " << option(_name="value").as<double>()
              << std::endl;

    LOG(INFO) << "proc " << Environment::worldComm().globalRank()
```

```
                <<" of "<< Environment::numberOfProcessors()
                << std::endl;

} // main
```

- We pass command line options using the [Boost Program Options](1) `po::` library. To add a new FEEL++ option, we must create a new FEEL++ `options_description`. You must add the default FEEL++ options and the new one that we choose here as a double value. Note that the default value will be assigned if not specified by the user.

- Then we initialize the environment variables through the FEEL++ `Environment` class (Check the Constructor prototype on the online documentation).

- we instantiate a new application. We specify the directory where to execute the program. That could be usefull for archiving your results.

- Finally, we save the results in a log file using the [google-glog library](2). As you can see, we save in this example our custom option value and the current processor number.

3.1.2 Compilation, execution, logs

To compile a tutorial, just use the GNU make command.

> make feelpp_doc_<appname>

where *appname* is the name of the application you wish to compile (here, `myapp`). Go to the execution directory as specified in the program, and execute it. You can change your option value.

> ./feelpp_doc_myapp [--value 6.6]

Now if you check the log,

> cat /tmp/<your login>/feelpp_doc_myapp/feelpp_doc_myapp.INFO

you should see your value and the processor number used to compute. You can run your application on several processors using MPI.

> mpirun -np 2 feelpp_doc_myapp

Note that there will be one log for each processor in that case.

3.1.3 Config files

A config file can be parsed to the program to profile your options. The default config paths are,

1. current dir

2. `$HOME/feel/config/`

3. `$INSTALL_PREFIX/share/feel/config/`

then you have to write inside one of these folders a file called `<app_name>.cfg` or `feelpp_<app_name>.cfg`. For example, our `myapp.cfg` would looks like,

```
    value=0.53
```

Note that you can specify the config file through the option `--config-file=<path>`

---

[1] http://www.boost.org/doc/libs/1_53_0/doc/html/program_options.html
[0] http://code.google.com/p/google-glog/

3.1.4   Initializing PETSc and Trilinos

PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for parallelism.

The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.

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

3.2   Mesh Manipulation

Feel++ provides some tools to manipulate mesh. Here is a basic example that shows you how to generate a mesh for a square geometry (source "doc/manual/tutorial/mymesh.cpp").

```cpp
int main( int argc, char** argv )
{
    // initialize Feel++ Environment
    Environment env( _argc=argc, _argv=argv,
                     _desc=feel_options(),
                     _about=about( _name="mymesh" ,
                                   _author="Feel++ Consortium",
                                   _email="feelpp-devel@feelpp.org" ) );

    // create a mesh with GMSH using Feel++ geometry tool
    auto mesh = unitSquare();

    // export results for post processing
    auto e = exporter( _mesh=mesh );
    e->save();

}   // main
```

As always, we initialise the FEEL++ environment (see section 3.1). The `unitSquare()` will generate a mesh for a square geometry. FEEL++ provides several functions to automate the GMSH mesh generation for different topologies. (`unitCircle()`, `unitCube()`, ...). These functions will create a geometry file *.geo* and a mesh file *.msh*. We can visualize them in GMSH.

> gmsh <entity_name>.msh

Finally we use the `exporter()` function to export the mesh for post processing. It will create by default a **Paraview** format file *.sos* and an **Ensight** format file *.case*.

> paraview <app_name>.sos

For advanced usage, there is the more generic `createGMSHMesh()` function which is useful for creating or loading an existing mesh or geometry (see section A.2.5 for a load example). Note that `unitSquare()` is just a particular case of `createGMSHMesh()`. FEEL++ provide useful tools to iterate on the mesh or some faces that we will see later. The process of the mesh creation is fully parallelized. You can as explained in section 3.1 run this example on several processors and visualise subregions with paraview.

3.3   Computing Integrals

You should be able to create a mesh now. If it is not the case, get back to the section 3.2. This part explains how to integrate on a mesh with FEEL++ (source "doc/manual/tutorial/myintegrals.cpp"). Let's consider the domain of the mesh,

$$\Omega = [0,1]^d = \{ x \in \mathbb{R}^d \,,\, x_i > 0 \; \sum_{i=1}^{d} x_i \leqslant 1 \} \in \mathbb{R}^d$$

Here, we want to integrate the following function,

$$f(x, y, z) = x^2 + y^2 + z^2 \tag{3.1}$$

on the whole domain Ω and on part of the boundary Ω. Take a look at the code.

```cpp
int
main( int argc, char** argv )
{
    // Initialize Feel++ Environment
    Environment env( _argc=argc, _argv=argv,
                     _desc=feel_options(),
                     _about=about( _name="myintegrals" ,
                                   _author="Feel++ Consortium",
                                   _email="feelpp-devel@feelpp.org" ) );

    // create the mesh (specify the dimension of geometric entity)
    auto mesh = unitHypercube<3>();

    // our function to integrate
    auto f = Px()*Px() + Py()*Py() + Pz()*Pz();

    // compute integral of f (global contribution)
    double intf_1 = integrate( _range = elements( mesh ),
                               _expr = f ).evaluate()( 0,0 );

    // compute integral of f (local contribution)
    double intf_2 = integrate( _range = elements( mesh ),
                               _expr = f ).evaluate(false)( 0,0 );

    // compute integral f on boundary
    double intf_3 = integrate( _range = boundaryfaces( mesh ),
                               _expr = f ).evaluate()( 0,0 );

    std::cout << "int global ; local ; boundary" << std::endl
              << intf_1 << ";" << intf_2 << ";" << intf_3 << std::endl;
}
```

To use the `integrate()` function, we have to precise the domain range. You can use,

- `elements()` to iterate on the whole mesh Ω,

- `boundaryfaces()` to iterate on the boundary $\partial\Omega$,

- `markedfaces()` to iterate on a choose face.

You have to specify the expression we wish to compute. FEEL++ provide a set of functions to write these expressions 4.1. The `evaluate()` function computes the integral on the global mesh. The `false` parameter limits the computation on the subregion owned by the processor. Note that FEEL++ computes automatically the quadrature and consider by default each non polynomial terms of the expression as a polynomial of degree 2. You can change it by passing a _quad parameter to the `integrate()` function which takes a `_Q<int order>` object as value. (refer to API documentation).

3.4 Function Spaces

```cpp
int main( int argc, char** argv )
{

    //Initialize Feel++ Environment
    Environment env( _argc=argc, _argv=argv,
                     _desc=feel_options(),
                     _about=about( _name="myfunctionspace",
                                   _author="Feel++ Consortium",
                                   _email="feelpp-devel@feelpp.org" )  );
```

```cpp
    auto g = sin( 2*pi*Px() )*cos( 2*pi*Py() )*cos( 2*pi*Pz() );

    // create the mesh
    auto mesh = unitSquare();

    // function space X_h using order 2 Lagrange basis functions
    auto Xh = Pch<2>( mesh );

    // elements of u,w ∈ X_h
    auto u = Xh->element( "u" );
    auto w = Xh->element( "w" );

    // build the interpolant
    u = vf::project( _space=Xh, _range=elements( mesh ), _expr=g );
    w = vf::project( _space=Xh, _range=elements( mesh ), _expr=idv( u )-g );

    // compute L2 norms
    double L2g = normL2( elements( mesh ), g );
    double L2uerror = normL2( elements( mesh ), ( idv( u )-g ) );
    std::cout << "||u-g||_0 = " << L2uerror/L2g << std::endl;

    // export for post-processing
    auto e = exporter( mesh );
    e->add( "g", u );
    e->add( "u-g", w );

    e->save();

}   // main
```

3.5   Linear Algebra

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

- Gmm

- Petsc[1]

- Trilinos[2]

3.5.1   *Choosing a linear algebra backend*

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

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

The backend provides an interface to solve

$$Ax = b \tag{3.2}$$

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 :

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

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**

---

[1] Petsc is a suite of data structures and routines for the scalable solution of scientific applications modeled by PDE available at http://www.mcs.anl.gov/petsc/petsc-as/

[2] The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for scientific problems. http://trilinos.sandia.gov/

```
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.

3.5.2 Solving

To solve the linear problem $Ax = b$, the backend provides a function `solve` with three required parameters

```
 solve(_matrix=A, _solution=x, _rhs=b)
```

where :

- the matrix A has a `sparse_matrix_ptrtype` type

- the solution x has a type `vector_type` or `vector_ptrtype`

- the second member vector b has a type `vector_ptrtype`

You can also add optional parameters like :

- a preconditioner : instead of solving $Ax = b$, we solve $P^{-1}Ax = P^{-1}b$. This method can be applied in iterative methods and permits to decrease the number of iterations in the resolution system

- a maximum number of iterations : this option is used with an iterative solving method

- a residual tolerance : the fraction $\dfrac{\|\ r^{(k)}\ \|}{\|\ r^{(0)}\ \|}$ is inferior to the residual tolerance with $r^{(k)} = b - Ax^{(k)}$ and $x^{(k)}$ the solution at the kth iteration

- a absolute tolerance : $\|\ r^{(k)}\ \|$ is inferior to the absolute tolerance

- a different tolerance : sometimes, the residue doesn't decrease continuously during the iterations. The difference between two plots doesn't have to exceed the parameter choosen for the difference tolerance.

- a boolean to use transpose matrix : instead of solving $Ax = b$, we solve $A^t x = b$. If A is defined and positive, $A^t = A$.

To have a view of the values of the optional parameters, see the following code :

```
BOOST_PARAMETER_MEMBER_FUNCTION(
        (solve_return_type),
         solve,
         tag,
         (required
        (matrix,(sparse_matrix_ptrtype))
        (in_out(solution),*(mpl::or_<boost::is_convertible<mpl::_,vector_type&>,
                                     boost::is_convertible<mpl::_,vector_ptrtype> >))
        (rhs,(vector_ptrtype)))
        (optional
          (prec,(sparse_matrix_ptrtype), matrix )
          (maxit,(size_type), 1000 )
          (rtolerance,(double), 1e-13)
          (atolerance,(double), 1e-50)
          (dtolerance,(double), 1e5)
          (reuse_prec,(bool), false )
          (transpose,(bool), false )
          )
      )
    {
```

The library `Boost::Parameters` allows you to enter parameters in the order you want. It supports deduced parameters, that is to say parameters whose identity can be deduced from their types.

3.6 Variational Formulation

3.6.1 *Principle*

A variational formulation of a problem is also called weak formulation. The key item is to bring a new function (called test function) and to integrate by parts. In that way we decrease the regularity constraint on our functions.

Let's consider the equation to solve with boundary conditions where $u \in \Omega$ is the unknown

$$
\begin{aligned}
-\Delta u &= f \\
u &= u_D \quad on \quad \Gamma_D \\
\nabla u.n &= g \quad on \quad \Gamma_N
\end{aligned}
\tag{3.3}
$$

$\Gamma = \Gamma_D \cup \Gamma_N$ is the border of Ω. By integrating by parts with a function v (called test function) supposed picewise regular, we obtain :

$$
\int_\Omega \nabla u \cdot \nabla v - \int_\Gamma (\nabla u \cdot n) v = \int_\Omega f v
$$

We have $u = u_D$ on Γ_D, we consequently take $v = 0$ on Γ_D and we got:

$$
\int_\Omega \nabla u \cdot \nabla v - \int_{\Gamma_N} g v = \int_\Omega f v \quad u \in \Omega, \forall v \in V
$$

where $V = \{v \in \Omega, v = 0 \text{ on } \Gamma_D\}$ with f and g which are known functions belonging to $C^0(\Omega)$. The test function v also has to be in $\mathbb{H}^1$. The condition $v = 0$ on Γ_D is often used but we obviously can impose more binding boundaries conditions on the test function. More generally, V_h represents the function test's space.

3.6.2 *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*

$$
- \Delta u = f \ in \ \Omega = [-1; 1]^2
\tag{3.4}
$$

with

$$
f = 2\pi^2 g
\tag{3.5}
$$

and g is the exact solution

$$
g = \sin(\pi x) \cos(\pi y)
\tag{3.6}
$$

The following boundary conditions apply

$$
u = g_{|x=\pm1}, \quad \frac{\partial u}{\partial n} = 0_{|y=\pm1}
\tag{3.7}
$$

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.

First one : strong Dirichlet conditions   The variational formulation reads as follows, we introduce the spaces

$$\mathcal{X} = \left\{ v \in H_1(\Omega) \text{ such that } v = g_{|x=-1, x=1} \right\} \tag{3.8}$$

and

$$\mathcal{V} = \left\{ v \in H_1(\Omega) \text{ such that } v = 0_{|x=-1, x=1} \right\} \tag{3.9}$$

We multiply (3.4) by $v \in \mathcal{V}$ then integrate over Ω and obtain

$$\int_\Omega -\Delta u v = \int_\Omega f v \tag{3.10}$$

We integrate by parts and reformulate the problem as follows:

Problem 2 *we look for $u \in \mathcal{X}$ such as*

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v \quad \forall v \in \mathcal{V} \tag{3.11}$$

In the present space setting (3.9) and boundary conditions (8.7), we have the boundary term from the integration by parts which is dropped being equal to 0

$$\int_{\partial\Omega} \frac{\partial u}{\partial n} v = 0, \tag{3.12}$$

recalling that

$$\frac{\partial u}{\partial n} \overset{\text{def}}{=} \nabla u \cdot n \tag{3.13}$$

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

$$\Omega = \cup_{e=1}^{N_{\text{el}}} \Omega^e \tag{3.14}$$

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

$$X_h = \left\{ v_h \in C^0(\Omega), \; v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e), \; v_h = g_{|x=-1, x=1} \right\} \tag{3.15}$$

and

$$V_h = \left\{ v_h \in C^0(\Omega), \; v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e), \; v_h = 0_{|x=-1, x=1} \right\} \tag{3.16}$$

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 $v \in V_h \subset \mathcal{V}$*

$$\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f v_h \tag{3.17}$$

Second one: 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 $v \in X_h$*

$$\int_\Omega \nabla u \cdot \nabla v + \int_{|x=-1, x=1} -\frac{\partial u}{\partial n} v - u \frac{\partial v}{\partial n} + \frac{\mu}{h} u v = \int_\Omega f v + \int_{|x=-1, x=1} -g \frac{\partial v}{\partial n} + \frac{\mu}{h} g v \tag{3.18}$$

where

$$X_h = \left\{ v_h \in C^0(\Omega), \; v_{h|\Omega^e} \in \mathbb{P}_N(\Omega^e) \right\} \tag{3.19}$$

In (7.3), g is defined by (8.6). 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

```cpp
    bool weak_dirichlet = this->vm()["weakdir"].template as<int>();
    value_type penaldir = this->vm()["penaldir"].template as<double>();
    value_type nu = this->vm()["nu"].template as<double>();
    std::string exact  = this->vm()[(boost::format("exact%1%D")%Dim).str()].template as<std::s
    std::string rhs  = this->vm()[(boost::format("rhs%1%D")%Dim).str()].template as<std::strin
    LOG(INFO) << "exact = "  << exact << "\n";

    auto vars=symbols<Dim>();
    ex ff;

    /*
     * if the right hand side is an empty string then use the string exact and
     * compute its laplacian to set the right hand side
     */
    if ( rhs.empty() )
        ff=-nu*laplacian(exact,vars);
    else
        ff = parse(rhs,vars);
    LOG(INFO) << "rhs="<< ff << "\n";
    auto g = expr(exact,vars);
    auto f = expr(ff,vars);
    auto gradg = expr<1,Dim,2>(grad(exact,vars), vars );
    LOG(INFO) << "grad(g)="<< grad(exact,vars) << "\n";

    // build Xh-interpolant of g
    gproj = project( _space=Xh, _range=elements( mesh ), _expr=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.

```cpp
    auto F = backend()->newVector( Xh );
    auto l = form1( _test=Xh, _vector=F );
    l = integrate( _range=elements( mesh ), _expr=f*id( v ) )+
        integrate( _range=markedfaces( mesh, "Neumann" ),
                    _expr=nu*gradg*vf::N()*id( v )
            );
```

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

$$\begin{aligned} \ell : \quad X_h \quad &\mapsto \quad \mathbb{R} \\ v_h \quad &\to \quad \ell(v_h) = \int_\Omega f v \end{aligned} \qquad (3.20)$$

which is represented algebraically in the code by the vector `F` using the argument `_vector`. The last argument `_init`, if set to **true**[3], 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`.

```cpp
    /** \code */
    auto D = backend()->newMatrix( _test=Xh, _trial=Xh  );
    /** \endcode */

    //! assemble ∫_Ω ν∇u · ∇v
    /** \code */
```

---

[3]It is set to **false** by default.

```
    auto a = form2( _test=Xh, _trial=Xh, _matrix=D );
    a = integrate( _range=elements( mesh ), _expr=nu*gradt( u )*trans( grad( v ) ) );
    /** \endcode */
```

Finally, we deal with the boundary condition, we implement both formulation described in appendix 8.2. For a *strong* treatment of the Dirichlet condition, we use the `on()` keyword of FEEL++ as follows

```
    a += on( _range=markedfaces( mesh, "Dirichlet" ),
             _element=u, _rhs=F, _expr=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 $x = \pm1$ 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:

```
    l += integrate( _range=markedfaces( mesh,"Dirichlet" ),
                    _expr=nu*g*( -grad( v )*vf::N()+penaldir*id( v )/hFace() ) );
```

```
    a += integrate( _range=markedfaces( mesh,"Dirichlet" ),
                    _expr= nu * ( -( gradt( u )*vf::N() )*id( v )
                                  -( grad( v )*vf::N() )*idt( u )
                                  +penaldir*id( v )*idt( u )/hFace() ) );
```

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

$$Du = F \tag{3.21}$$

where the `solve` function is implemented as follows

```
    backend( _rebuild=true )->solve( _matrix=D, _solution=u, _rhs=F );
```

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

$$\|u - u_h\|_{L_2} \;=\; \sqrt{\int_\Omega (u - u_h)^2} = \sqrt{\int_\Omega (g - u_h)^2} \tag{3.22}$$

where u is the exact solution and is equal to g and u_h is the numerical solution of the problem (3.4) and the components of u_h in the P_2 Lagrange basis are given by solving (8.5).

The code reads

```
    double L2error =normL2( _range=elements( mesh ),_expr=( idv( u )-g ) );

    LOG(INFO) << "||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 3.3 displays the results using Paraview.

Let's try a parallel computation using 3 processors. The following figures display the results using Paraview.

Remark 1 *You can modify the domain size in order to obtain a clearer figure by adding diffrent values to* `_xmin` `_ymin` `_xmax` *and* `_ymax` *in the definition of the mesh in* `laplacian.cpp`
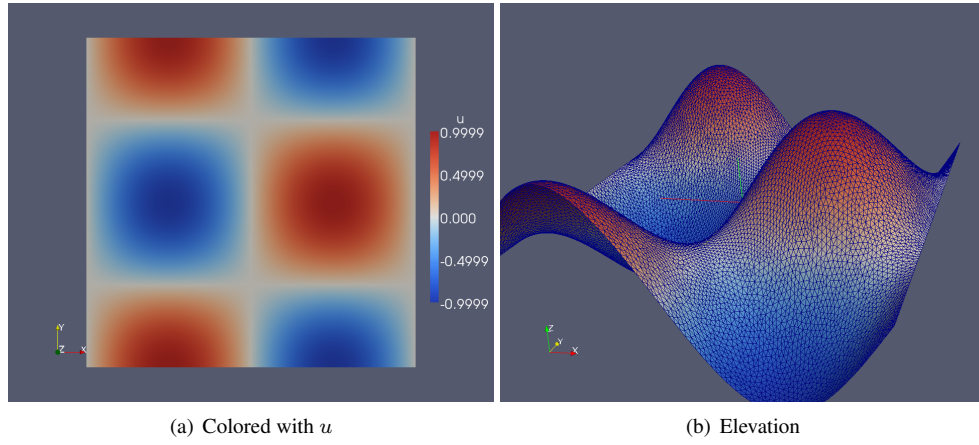
(a) Colored with u

(b) Elevation

Figure 3.1: Solution of problem 4

(a) domain 1

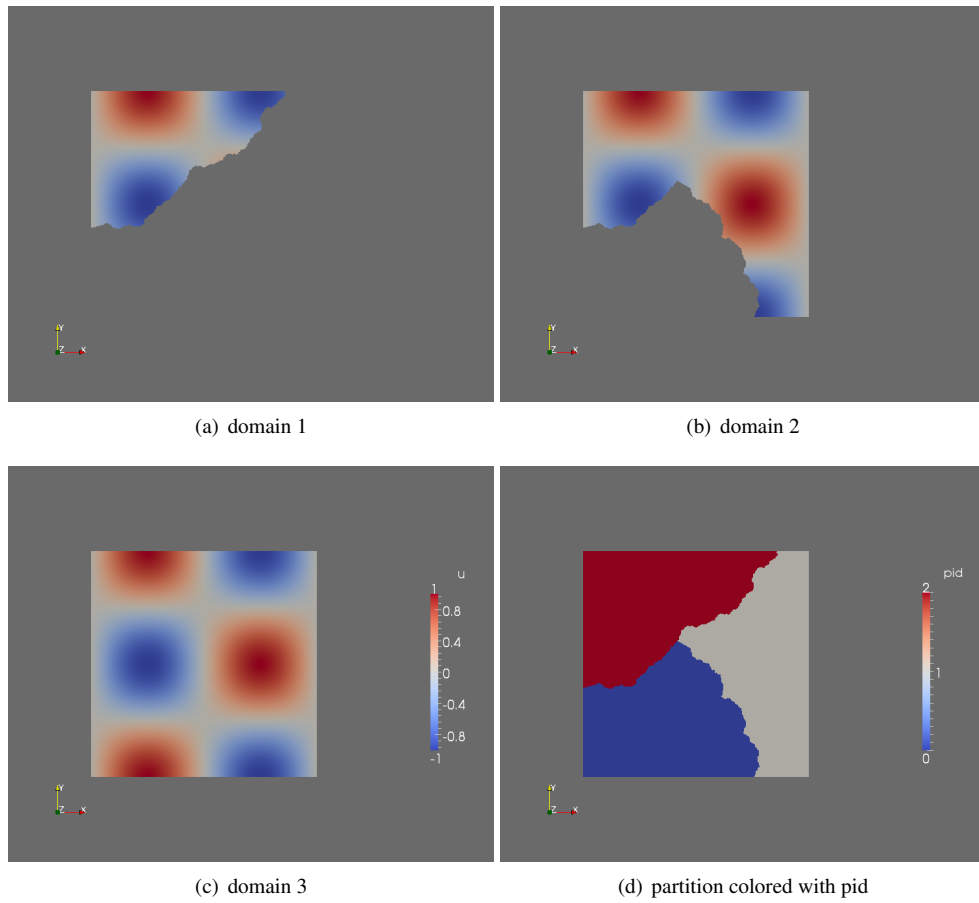(b) domain 2

(c) domain 3

(d) partition colored with pid

Figure 3.2: Solution of the laplacian problem

3.6.3 Mixed formulation: the Stokes case

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*

$$-\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \quad and \quad \nabla\cdot\mathbf{u} = 0, \quad in\ \Omega = [-1;1]^2 \tag{3.23}$$

with

$$\mathbf{f} = \mathbf{0} \tag{3.24}$$

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

$$\mathbf{u} = \mathbf{1}_{|y=1}, \quad \mathbf{u} = \mathbf{0}_{|\partial\Omega\setminus\{(x,y)\in\Omega|y=1\}} \tag{3.25}$$

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.*

$$\int_\Omega p = 0 \tag{3.26}$$

The problem 5 now reads

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

$$-\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \quad, \quad \nabla\cdot\mathbf{u} + \lambda = 0, \quad and \quad \int_\Omega p = 0, \quad in\ \Omega = [-1;1]^2 \tag{3.27}$$

with

$$\mathbf{f} = \mathbf{0} \tag{3.28}$$

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

$$\mathbf{u} = \mathbf{1}_{|y=1}, \quad \mathbf{u} = \mathbf{0}_{|\partial\Omega\setminus\{(x,y)\in\Omega|y=1\}} \tag{3.29}$$

The functional framework is as follows, we look for $\mathbf{u}$ in $H_0^1(\Omega)$ and p in $L_0^2(\Omega)$. We shall not seek p in $L_0^2(\Omega)$ 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 (3.26).

Denote $\mathcal{X} = H_0^1(\Omega) \times L^2(\Omega) \times \mathbb{P}_0(\Omega)$, the variational formulation reads we look for $(\mathbf{u}, p, \lambda) \in \mathcal{X}$ for all $(\mathbf{v}, q, \nu) \in \mathcal{X}$

$$\int_\Omega \mu\nabla\mathbf{u} : \nabla\mathbf{v} + \nabla\cdot\mathbf{v}\,p + \nabla\cdot\mathbf{u}\,q + q\lambda + p\nu \ = \ \int_\Omega \mathbf{f}\cdot\mathbf{v} \tag{3.30}$$

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 $(\mathbf{v}_h, q_h, \nu_h) \in \mathcal{X}_h$

$$\int_{\Omega_h} \mu\nabla\mathbf{u}_h \cdot \nabla\mathbf{v}_h + \nabla\cdot\mathbf{v}_h\,p_h + \nabla\cdot\mathbf{u}_h\,q_h + p_h\nu_h + q_h\lambda_h = \int_{\Omega_h} \mathbf{f}\cdot\mathbf{v}_h \tag{3.31}$$

The formulation (3.31) leads to a linear system of the form

$$\underbrace{\begin{pmatrix} A & B & 0 \\ B^T & 0 & C \\ 0 & C^T & 0 \end{pmatrix}}_{\mathcal{A}} \underbrace{\begin{pmatrix} \mathbf{u}_h \\ p_h \\ \lambda_h \end{pmatrix}}_{\mathcal{U}} = \underbrace{\begin{pmatrix} F \\ 0 \\ 0 \end{pmatrix}}_{\mathcal{F}} \tag{3.32}$$

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 symmetric 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 3.6.2. 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 $\mathbb{P}_0(\Omega)$.

```cpp
    typedef Lagrange<2, Vectorial> basis_u_type;
    typedef Lagrange<1, Scalar> basis_p_type;
    typedef Lagrange<0, Scalar> basis_l_type;
    // use lagrange multipliers to ensure zero mean pressure
#if defined( FEELPP_USE_LM )
    typedef bases<basis_u_type,basis_p_type, basis_l_type> basis_type;
#else
    typedef bases<basis_u_type,basis_p_type> basis_type;
#endif
```

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:

```cpp
    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.

```cpp
    typedef space_type::element_type element_type;
```

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

```cpp
```

They will serve in the definition of the variational formulation. We can now start assemble the various terms of the variational formulation (3.31). First we define some viscous stress tensor, $\tau(\mathbf{u}) = \nabla\mathbf{u}$, associated with the trial and test functions respectively

```cpp
    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 $\bar{\sigma}(\mathbf{u}, p) = -p\mathbb{I} + 2\mu\tau(\mathbf{u})$:

```cpp
    // 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.

```cpp
    auto stokes = form2( _test=Xh, _trial=Xh, _matrix=D );

    stokes += integrate( elements( mesh ), mu*inner( deft,def ) );
    LOG(INFO) << "chrono mu*inner(deft,def): " << chrono.elapsed() << "\n";
    chrono.restart();
    stokes +=integrate( elements( mesh ), - div( v )*idt( p ) + divt( u )*id( q ) );
    LOG(INFO) << "chrono (u,p): " << chrono.elapsed() << "\n";
    chrono.restart();
#if defined( FEELPP_USE_LM )
    stokes +=integrate( elements( mesh ), id( q )*idt( lambda ) + idt( p )*id( nu ) );
    LOG(INFO) << "chrono (lambda,p): " << chrono.elapsed() << "\n";
    chrono.restart();
#endif

    stokes +=integrate( boundaryfaces( mesh ), -inner( SigmaNt,id( v ) ) );
    stokes +=integrate( boundaryfaces( mesh ), -inner( SigmaN,idt( u ) ) );
    stokes +=integrate( boundaryfaces( mesh ), +penalbc*inner( idt( u ),id( v ) )/hFace() );
    LOG(INFO) << "chrono bc: " << chrono.elapsed() << "\n";
    chrono.restart();
```

The figure 3.3 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
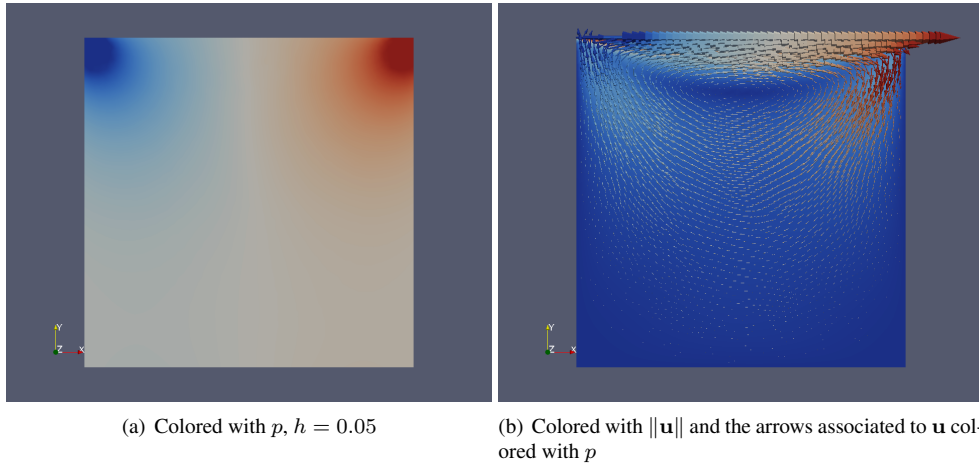
Figure 3.3: Solution of problem 5

CHAPTER 4

Feel++ Language Keywords

By Christophe Prud'homme

Chapter ref: **[cha:appendix-feel]**

4.1 Keywords

One of FEEL++ assets is it finite element embedded language. 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.

Here are 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

and here is the table which gathers all tools you may need:

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

Keyword	Math object	Description	Rank	$M \times N$
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
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,	$\begin{pmatrix} m_{11} & m_{12} & ... \\ m_{21} & m_{22} & ... \\ \vdots & & \end{pmatrix}$	$M \times N$ matrix	2	$M \times N$
m_12,...)		entries being expressions		
vec<M>(v_1,	$(v_1, v_2, ...)^T$	column vector with M rows	1	$M \times 1$
v_2,...)		entries being expressions		
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]

[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	$M \times N$
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	$m = 2, 1 \times$
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	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
averaget(f)	$f = \frac{1}{2}(f_0 + f_1)$	average of trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
averagev(f)	$f = \frac{1}{2}(f_0 + f_1)$	average of function evaluation	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftface(f)	f_0	left test function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftfacet(f)	f_0	left trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
leftfacev(f)	f_0	left function evaluation	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightface(f)	f_1	right test function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightfacet(f)	f_1	right trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
rightfacev(f)	f_1	right function evaluation	$\mathrm{rank}(f(\overrightarrow{x}))$	$m = n, n \times$
maxface(f)	$\max(f_0, f_1)$	maximum of right and left test function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
maxfacet(f)	$\max(f_0, f_1)$	maximum of right and left trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
maxfacev(f)	$\max(f_0, f_1)$	maximum of right and left function evaluation	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
minface(f)	$\min(f_0, f_1)$	minimum of right and left test function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
minfacet(f)	$\min(f_0, f_1)$	minimum of right and left trial function	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
minfacev(f)	$\min(f_0, f_1)$	minimum of right and left function evaluation	$\mathrm{rank}(f(\overrightarrow{x}))$	$m \times p$
-	$-g$	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		

Keyword	Math object	Description	Rank	$M \times N$
||	f or g	element wise logical or		

4.2 Operators

4.2.1 Integrals

Thank to its finite element embedded language, FEEL++ has its owned `integrate()` function, which can be written for example :

```
integrate( _range= elements(mesh), _expr= gradt(T)*trans(grad(v)) );
```

please notice that the order of the parameter is not important, these are `boost` parameters, so you can enter them in the order you want. To make it clear, there are two required parameters and 2 optional and they of course can be entered in any order provided you give the parameter name. If you don't provide the parameter name (that is to say `_range=` or the others) they must be entered in the order they are described below.

The required parameters are

- `_range` = domain of integration

- `_expr` = integrand expression

The optional parameters are

- `_quad` = quadrature to use instead of the default one, wich means `_Q<integer>()` where the integer is the polynomal order to integrate exactly

- `_geomap` = type of geometric mapping to use, that is to say :

    - `GEOMAP_HO` = high order approximation (same of the mesh)

    - `GEOMAP_OPT` = optimal approximation: high order on boundary elements, order 1 in the interior

    - `GEOMAP_O1` = order 1 approximation

4.2.2 Projections

It is also possible to make projections with the library, the interface is as follow :

```
project( _range, _space, _expr, _geomap );
```

where

- `_space` is the space in which lives the projected expression, it should be a nodal function space

- `_expr` the expression to project

- `_range` is the domain for the projection (optional, default: all elements from `space->mesh()`)

- `_geomap` is the type of geometric mapping approximation (optional, default = `GEOMAP_HO`)

- `_accumulate` (optional, default = false)

4.2.3 Meshes

FEEL++ enables full different ways to interact with the mesh on which you want to work. Mainly with the function `integrate`, the various keywords we have established will make your program's code easier. The interoperability between FEEL++ and GMSH is huge and provides various access to any point, item, domain or almost anything you want in a mesh. The access to different items of a mesh is possible thanks to the filters which enable the access of only a mesh's part. Theses helpfull keywords are coded in `feel/feelmesh/filters.hpp`, we are here going to describe most of them.

To access one particular part of a mesh, you can use :

- `elements(mesh)` corresponds to all the elements of a mesh

- `markedelements(mesh, id)` corresponds to the precise element defined by the id. It can be any element (line, surface, domain, and so on).

- `faces(mesh)` corresponds to all the faces of the mesh.

- `markedfaces(mesh)` corresponds to all the faces of the mesh which are marked.

- `boundaryfaces(mesh)` corresponds to all elements that own a topological dimension one below the mesh. For example, if you mesh is a 2D one, `boundaryfaces(mesh)` will return all the lines (because of dimension $2-1 = 1$). These elements which have one dimension less, are corresponding to the boundary faces.

- `internalelements(mesh)` corresponds to all the elements of the mesh which are stricly within the domain that is to say they do not share a face with the boundary.

- `boundaryelements(mesh)` corresponds to all the elements of the mesh which share a face with the boundary of the mesh.

- `edges(mesh)` corresponds to all the edges of the mesh.

- `boundaryedges(mesh)` corresponds to all boundary edges of the mesh.

where id is the element's identifier : thanks to GMSH, this identifier can be an integer or a string, it depends on the identifier your have or you gave in the mesh .geo file.

Part II

Learning by Examples

CHAPTER 5

Non-Linear examples

By Christophe Prud'homme

Chapter ref: **[cha:non-linear-ex]**

5.1 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

```cpp
#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
  }
```

```cpp
  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.

5.1.1   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

$$- \Delta u + u^\lambda = f, \quad u = 0 \text{ on } \partial\Omega. \tag{5.1}$$

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

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

5.1.2   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

$$- \Delta u + \lambda e^u = f, \quad u = 0 \text{ on } \partial\Omega \tag{5.2}$$

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/manual/bratu.cpp` for an implementation of this problem.

Baptiste Morin, Christophe Prud'homme

CHAPTER 6

Heat sink

By Baptiste Morin, Christophe Prud'homme

Chapter ref: **[cha:heatsink]**

This problem considers the performance of a heat sink designed for the thermal management of high-density electronic components. The heat sink is comprised of a base/spreader which in turn supports a number of plate fins exposed to flowing air. We model the flowing air through a simple convection heat transfer coefficient. From the engineering point of view, this problem illustrates the application of conduction analysis to an important class of cooling problems: electronic components and systems.

Our interest is in the conduction temperature distribution at the base of the spreader. The target is to study how the heat transfer occures with different parameters on our heat sink. The heat generated by high-density electronic components is such that it's very expensive to cool large structures (data center). The cooling optimization is consequent in the run for decreasing operating costs.

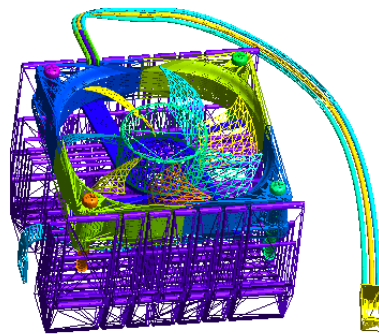A classical thermal CPU cooler looks like this

Figure 6.1: Mesh of a classical CPU cooler

We are here going to describe how it is theorically working and how it is impleted with FEEL++.

45

6.1 Problem description

6.1.1 Domain

We consider here a classical "radiator" which is a CPU heat sink. Those types of coolers are composed with a certain number of plate fins exposed to flowing air or exposed to a ventilator. Regarding the periodicity and geometry of our concern, we can make our study on a characteristic element of the problem : a half cell of the heat sink single thermal fin with its spreader at the basis. Let's take a look at the geometry of our problem :
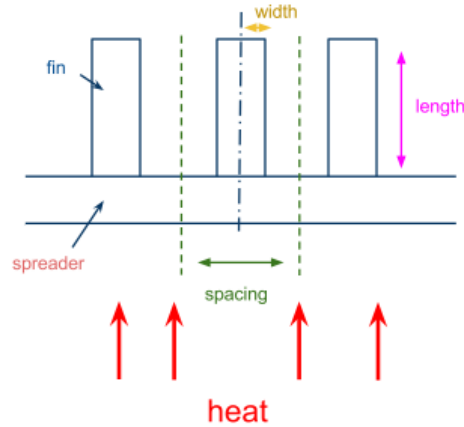
Figure 6.2: Geometry of heat sink

Our study is avaible in 2 or 3 dimensions, depending on the application's parameters. You'll see later how to work with it. Let's see on which meshes we are working on :

Figure 6.3: 2D mesh
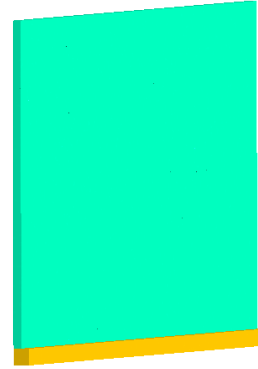
Figure 6.4: 3D mesh

6.1.2 Inputs

The implementation of thoses parameters is described in the section 6.3.1.

Material

Here the material parameter can be described with furthers parameters. We have, with $i = 1$ for the fin and $i = 2$ for the base :

- the thermal conductivity κ_i

- the material's density ρ_i

- the heat capacity of the material C_i

The term $\rho_i C_i$ corresponds to the heat volumetric capacity. In that way, we make possible the construction of a heat sink with 2 different materials. Here is a list of the well-known ones, ρ and C are gave at $298K$:

| Material	Thermal conductivity (κ in $W.m^{-1}.K^{-1}$)	Density (ρ in $kg.m^{-3}$)	Heat Capacity (C in $J.kg^{-1}.K^{-1}$)
Aluminium	180 (alloys) or 290 (pure)	2700	897
Copper	386	8940	385
Gold	314	19320	129
Silver	406	10500	233

Physical

- Depth
  This parameter is only to take into account for the 3D simulation. It represents the depth of the caracteristical heat sink and is called `depth` in the application.

- Length
  You can also parameterize the length of the fin. This one is called `L` in the application's parameters, its dimension is the meter.

- Width
  Typically, this parameter is linked with constructor's standards. This parameter is called `width` in the application's implementation.

Thermal

- Heat flux
  It represents the heat flux brought by the electronic component at the bottom of the base. Here it's typically the heat brought by the processor.

- Thermal coefficient
  The thermal coefficient h named $therm_c oeff$ in the application is representative of the heat transfer between the fin and the air flow.

- Ambien temperature
  This parameter called T_{amb} represents the temperature around the heat sink at the beginning. That means the ambient temperature before the computer is turned on.

Summary table

The following table displays the various fixed and variables parameters of this application.

Name	Description	Nominal Value	Range	Units
BDF parameters				
$time - initial$	begining	0		
$time - final$	end	50	$]0, 1500]$	
$time - step$	time step	0.1	$]0, 1[$	
$steady$	steady state	0	$\{0, 1\}$	
$order$	order	2	$[0, 4]$	
Physical parameters				
L	fin's length	$2 \cdot 10^{-2}$	$[0.02, 0.05]$	m
$width$	fin's width	$5 \cdot 10^{-4}$	$[10^{-5}, 10^{-4}]$	m
$deep$	heat sink depth	0	$[0, 7 \cdot 10^{-2}]$	m
Mesh parameter				
$hsize$	mesh's size	10^{-4}	$[10^{-5}, 10^{-3}]$	
Fin Parameters				
κ_f	thermal conductivity	386	$[100, 500]$	$W \cdot m^{-1} \cdot K^{-1}$
ρ_f	material density	8940	$[10^3, 12 \cdot 10^3]$	$kg \cdot m^{-3}$
C_f	heat capacity	385	$[10^2, 10^3]$	$J \cdot kg^{-1} \cdot K^{-1}$
Base/spreader Parameters				
κ_s	thermal conductivity	386	$[100, 500]$	$W \cdot m^{-1} \cdot K^{-1}$
ρ_s	material density	8940	$[10^3, 12 \cdot 10^3]$	$kg \cdot m^{-3}$
C_s	heat capacity	385	$[10^2, 10^3]$	$J \cdot kg^{-1} \cdot K^{-1}$
Heat Parameters				
T_{amb}	ambient temperature	300	$[300, 310]$	K
$heat\_flux$	heat flux Q	10^6	$[0, 10^6]$	$W \cdot m^{-3}$
$therm\_coeff$	thermal coefficient h	10^3	$[0, 10^3]$	$W \cdot m^{-2} \cdot K^{-1}$

Table 6.1: Table of fixed and variable parameters

6.2 Theory

6.2.1 Figure

The global domain is $\Omega = \Omega_1 \cup \Omega_2$ where Ω_1 is the fin's domain and Ω_2 the spreader's domain. We note $\partial\Omega$ the border of the domain Ω. The physical lines we are using will be noted as Γ_i such as described above. The following figure describes the parameters and the geometry we are using in the equations to solve our 3D issue : The following figures describe the parameters and the geometry we are using in the equations to solve our 2D or 3D issue :
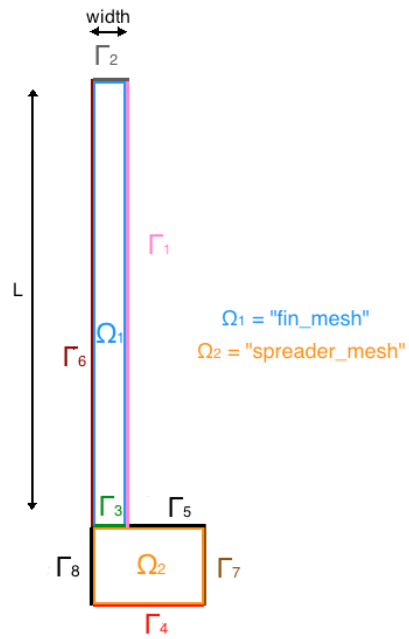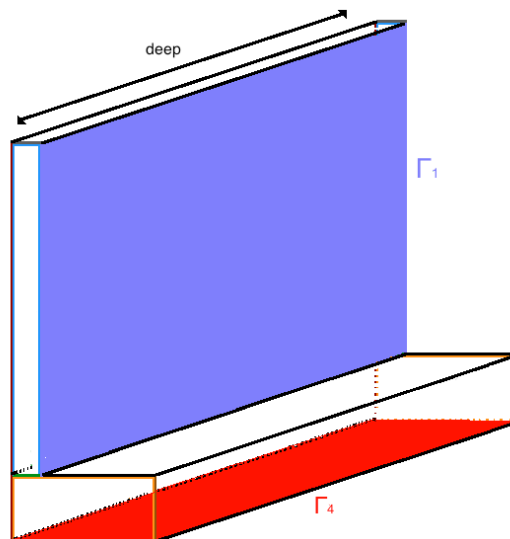
Figure 6.5: 2D geometry details

Figure 6.6: 3D geometry details

6.2.2 Equations

Our concern satisfies the heat equation which reads

$$\sum_{i=1}^{2} \kappa_i \Delta T - \rho_i C_i \frac{\partial T}{\partial t} = 0 \tag{6.1}$$

$$\kappa_1 \frac{\partial T}{\partial n} = 0 \quad on \quad \Gamma_2 \quad and \quad \Gamma_6 \tag{6.2}$$

$$\kappa_2 \frac{\partial T}{\partial n} = 0 \quad on \quad \Gamma_5, \Gamma_7 \quad and \quad \Gamma_8 \tag{6.3}$$

$$\kappa_1 \frac{\partial T}{\partial n} = -h(T - T_{amb}) \quad on \quad \Gamma_1 \tag{6.4}$$

$$\kappa_2 \frac{\partial T}{\partial n} = Q(1 - e^{-t}) \quad on \quad \Gamma_4 \tag{6.5}$$

$$T_{|\Omega_1} = T_{|\Omega_2} \quad on \quad \Gamma_3 \tag{6.6}$$

$$\kappa_1 \nabla T \cdot n = \kappa_2 \nabla T \cdot n \quad on \quad \Gamma_3 \tag{6.7}$$

with $i = 1$ for the fin and $i = 2$ for the base and where κ_i is the thermal conductivity, ρ_i is the material's density ($kg.m^{-3}$ in the SI unit), C_i the heat capacity and T the temperature at a precise point (in 2D or 3D). To see how it has been coded, you can read 6.3.3.

6.2.3 Boundary conditions

The problem requieres that the temperature and heat flux are continute on Γ_3. Considering the problem's geometry, we also impose zero heat flux on the vertical surfaces of the spreader. Let's detail the conditions we have imposed :

- Homogeneous Neumann condition (6.2) and (6.3) : it represents the fact that the heat flux is only vertical (for Γ_6 and Γ_7) or the fact that the heat flux is only provided by Γ_4 (for Γ_2 and Γ_5).

- Homogeneous Neumann condition (6.4) : it imposes that the heat flux is brought by this surface (it mathematically represents that the heat sink is placed on the heat source).

- Non-homogeneous Neumann condition (6.5) : this boundary condition represents the transient state for the heat transfer calculation.

- Temperature continuity (6.6) : it imposes that the temperature is continute at the interface between the two materials (if there are two materials, we can also have the same one for the two pieces).

- Heat flux continuity (6.7) : it represents that the heat flux is continute at the interface between the two materials. Literally, it means that the two flows offset each other.

Theses conditions have been coded as explained in the section 6.3.3.

6.2.4 Finite Element Method

Let's apply the method to our concern, we introduce the test function v and we integrate the main equation, which reads now as :

$$\sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} - \kappa_i \int_{\Omega_i} v \Delta T = 0 \tag{6.8}$$

We integrate by parts, which leads to :

$$\sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \kappa_i \int_{\partial \Omega_i} (\nabla T \cdot n) v = 0 \tag{6.9}$$

then, by decomposing the borders $\partial \Omega_i$, we obtain :

$$-\kappa_1 \int_{\Gamma_1} (\nabla T \cdot n) v - \kappa_2 \int_{\Gamma_4} (\nabla T \cdot n) v - \kappa_1 \int_{\Gamma_{2,6}} (\nabla T \cdot n) v - \kappa_2 \int_{\Gamma_{5,7,8}} (\nabla T \cdot n) v \quad +$$

$$\sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \kappa_i \int_{\partial \Omega_i \cap \Gamma_3} (\nabla T \cdot n) v = 0 \tag{6.10}$$

Now, we apply the conditions (6.2), (6.3), (6.4) and (6.5) which brings us to :

$$\int_{\Gamma_1} h v (T - T_{amb}) - \int_{\Gamma_4} v Q (1 - e^{-t}) + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \underbrace{\kappa_i \int_{\partial \Omega_i \cap \Gamma_3} (\nabla T \cdot n) v}_{=0 \text{ thanks to } 6.7} = 0 \tag{6.11}$$

Now we apply the boundary conditions (6.7) which results in :

$$h \int_{\Gamma_1} v (T - T_{amb}) - \int_{\Gamma_4} v Q (1 - e^{-t}) + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = 0 \tag{6.12}$$

We can now start to transform the equation by puting in the right hand the known terms :

$$h \int_{\Gamma_1} v T + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} v Q (1 - e^{-t}) + h T_{amb} \int_{\Gamma_1} v \tag{6.13}$$

We discretize $\dfrac{\partial T}{\partial t}$ where δt is the time step, such as:

$$h \int_{\Gamma_1} v T + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{T^{n+1} - T^n}{\delta t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} v Q (1 - e^{-t}) + h T_{amb} \int_{\Gamma_1} v \tag{6.14}$$

Finally we obtain :

$$h \int_{\Gamma_1} v T + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{T^{n+1}}{\delta t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} v Q (1 - e^{-t}) + h T_{amb} \int_{\Gamma_1} v + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v \frac{T^n}{\delta t} \tag{6.15}$$

This is that equation which is implemented in the application `feel_heatsink`.

6.3 Implementation

6.3.1 *Application parameters*

The parameters of the application are implemented such as

```cpp
inline
Feel::po::options_description
makeOptions()
{
Feel::po::options_description heatsinkoptions("heatsink options");
heatsinkoptions.add_options()
// mesh parameters
("hsize", Feel::po::value<double>()->default_value( 0.1 ),
 "first h value to start convergence")
("L", Feel::po::value<double>()->default_value( 0.03 ),
 "dimensional length of the sink (in meters)")
("width", Feel::po::value<double>()->default_value( 0.0005 ),
 "dimensional width of the fin (in meters)")

// 3D parameter
("deep", Feel::po::value<double>()->default_value( 0 ),
 "depth of the mesh (in meters) only in 3D simulation")

// thermal conductivities parameters
("kappa_s", Feel::po::value<double>()->default_value( 386 ),
 "thermal conductivity of the base spreader in SI unit W.m^{-1}.K^{-1}")
("kappa_f", Feel::po::value<double>()->default_value( 386 ),
 "thermal conductivity of the fin in SI unit W.m^{-1}.K^{-1}")

// density parameter
("rho_s", Feel::po::value<int>()->default_value( 8940 ),
 "density of the spreader's material in SI unit kg.m^{-3}")
("rho_f", Feel::po::value<int>()->default_value( 8940 ),
 "density of the fin's material in SI unit kg.m^{-3}")

// heat capacities parameter
("c_s", Feel::po::value<double>()->default_value( 385 ),
 "heat capacity of the spreader's material in SI unit J.kg^{-1}.K^{-1}")
("c_f", Feel::po::value<double>()->default_value( 385 ),
 "heat capacity of the fin's material in SI unit J.kg^{-1}.K^{-1}")

// physical coeff
("therm_coeff", Feel::po::value<double>()->default_value(50),
 "thermal coefficient")
("Tamb", Feel::po::value<double>()->default_value(300),
 "ambiant temperature")
("heat_flux", Feel::po::value<double>()->default_value(1e6),
 "heat flux generated by CPU")

("steady", Feel::po::value<bool>()->default_value(false),
 "if true : steady else unsteady")

// export
("export-matlab", "export matrix and vectors in matlab" );

return heatsinkoptions.add( Feel::feel_options() );
}
```

6.3.2 *Surfaces*

To be able to calculate the surfaces in further dimension without changing the code, we have given the same names for the faces we were interested in. In 2D Γ_i represents a line whereas in 3D it represents a surface. The calculation of those surfaces which makes possible the calculation of averages temperature is as follow :

```cpp
surface_base =
 integrate( _range= markedfaces(mesh,"gamma4"), _expr= cst(1.)).evaluate()(0,0);
```

```
surface_fin =
 integrate( _range= markedfaces(mesh,"gamma1"), _expr=cst(1.)).evaluate()(0,0);
```

6.3.3   Equations

First we start by calculate the non-steady state which means that we integrate all the time-independant terms, which is done with :

```
/*
 * Right hand side construction (steady state)
 */
form1( _test=Xh, _vector=F, _init=true ) =
 integrate( _range= markedfaces(mesh, "gamma1"), _expr= therm_coeff*Tamb*id(v));

/*
 * Left hand side construction (steady state)
 */
form2( Xh, Xh, D, _init=true ) =
integrate( _range= markedelements(mesh,"spreader_mesh"),
                _expr= kappa_s*gradt(T)*trans(grad(v)) );

form2( Xh, Xh, D) +=
integrate( _range= markedelements(mesh,"fin_mesh"),
                _expr= kappa_f*gradt(T)*trans(grad(v)) );

form2 (Xh, Xh, D) +=
integrate( _range= markedfaces(mesh, "gamma1"),
                _expr= therm_coeff*idt(T)*id(v));

form2(Xh, Xh, D) +=
  integrate( _range=markedelements(mesh, "spreader_mesh"),
                _expr=rho_s*c_s*idt(T)*id(v)*M_bdf->polyDerivCoefficient(0) )
  + integrate( _range=markedelements(mesh, "fin_mesh"),
                _expr=rho_f*c_f*idt(T)*id(v)*M_bdf->polyDerivCoefficient(0) );
```

Then, to compute the transient state, which means time dependant terms, you have to initialize the temperature (which is initialized as T_{amb} on X_h space) and create a new vector F_t which corresponds to the time dependent term. The code is as follow :

```
T = vf::project( _space=Xh, _expr=cst(Tamb) );
M_bdf->initialize(T);
auto Ft = M_backend->newVector( Xh );

for ( M_bdf->start(); M_bdf->isFinished()==false; M_bdf->next() )
{
        // update right hand side with time dependent terms
        auto bdf_poly = M_bdf->polyDeriv();
        form1( _test=Xh, _vector=Ft ) =
            integrate( _range=markedelements(mesh, "spreader_mesh"),
                        _expr=rho_s*c_s*idv(bdf_poly)*id(v)) +
            integrate( _range=markedelements(mesh, "fin_mesh"),
                        _expr=rho_f*c_f*idv(bdf_poly)*id(v) );

        form1( _test=Xh, _vector=Ft ) +=
            integrate( _range= markedfaces(mesh,"gamma4"),
                        _expr= heat_flux*(1-exp(-M_bdf->time()))*id(v) );

        // add contrib from time independent terms
        Ft->add( 1., F );

        // solve
        M_backend->solve( _matrix=D, _solution=T, _rhs=Ft );

        // both average temperatures
        Tavg = integrate( _range=markedfaces(mesh,"gamma4"),
                        _expr=(1/surface_base)*idv(T) ).evaluate()(0,0);

        Tgamma1 = integrate( _range=markedfaces(mesh,"gamma1"),
                        _expr=(1/surface_fin)*idv(T) ).evaluate()(0,0);
```

```
        // export results
        out << M_bdf->time() << " " << Tavg << " " << Tgamma1 << "\n";

        this->exportResults( M_bdf->time(), T );

}
```

6.3.4 Outputs

As you can see in the equation's implementation above, there are two ouputs :

- GMSH format : this file contains the entire mesh and the temperatures associated to each degrees of freedom of the mesh. To open it, you juste have to do as you always do with GMSH : `gmsh heatsink-1_0.msh`. You will obtain the figure with the different temperatures, you are now able to click on "play" with its significative logo and admire the evolution

- `averages` file : this file is completed at each time step, each line contains the current time, the average temperature on Γ_4 (surface where is the contact between the heat sink and the heat source) and the average temperature on Γ_1. To analyze this file, we recommend you to work with OCTAVE which is an open-source software similar to MATLAB. If it is installed, open a command line and go to `~/feel/heatsink/Simplex_*.*.*/0.000*/` and try :

```
> octave
octave:1> M=load('averages');
octave:2> plot(M(:,1),M(:,2))
octave:3> plot(M(:,1),M(:,3))
octave:4> plot(M(1:70,1),M(1:70,2))
octave:5> plot(M(1:70,1),M(1:70,3))
```

The 4th and 5th lines are here to observe the transient state.

6.4 Use cases

6.4.1 How to use it ?

To make easier the use of this application, we recommand you to use the configurations files. This is the fastest way : to do it, you juste have to create the file `heatsink.cfg` and place it in the same directory that your application's executable.
We have created 3 typical `cfg` files such as :

```
# file heatsink_1.cfg
# spreader and fin in copper
# 2D simulation
hsize=1e-4

kappa_s=386 # W/m/K
c_s=385
rho_s=8940

kappa_f=386 # W/m/K
c_f=385 #J/kg/K
rho_f=8940

L=15e-3
width=5e-4

therm_coeff=1000   #W/(m2K)
heat_flux=1e6

[bdf]
order=2
time-step=0.05
time-final=100
steady=0
```

```
[exporter]
format=gmsh
```

```
# file heatsink_3.cfg
# spreader in copper
# fin in aluminium
# 3D simulation
hsize=3e-4
kappa_s=386 # W/m/K
c_s=385
rho_s=8940

kappa_f=386 # W/m/K
c_f=385 #J/kg/K
rho_f=8940

L=15e-3
width=5e-4
deep=4e-2

therm_coeff=1000   #W/(m2K)
heat_flux=1e6

[bdf]
order=2
time-step=0.05
time-final=100
steady=0

[exporter]
format=gmsh
```

This file is the only modification you will have to bring to the application, in that way you won't have to compile each time the files (except for `heatsink.cpp` if you want to increase the order and/or the dimension, in that case you'ill have to modify this parameter at then end of the file in the `main` method).

6.4.2   Results

2D cases

Here are some results of the 2D simulations considering different configurations files. The figures have been extracted thanks to GMSH and OCTAVE :

Figure 6.7: `heatsink_1.cfg` : steady state, spreader and fin in copper, $Q = 1e6$ and $h = 1e3$
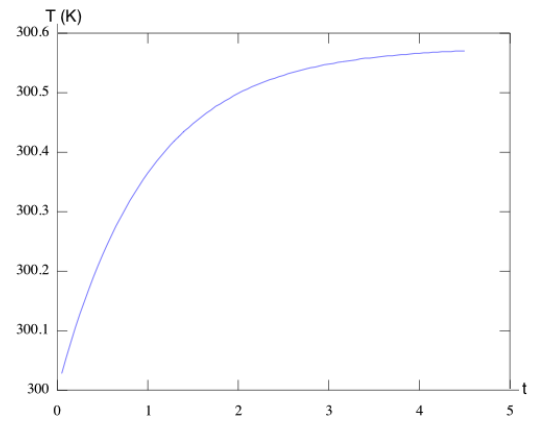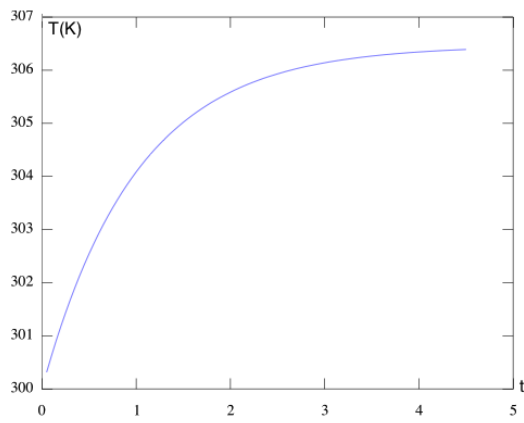
Figure 6.8: `heatsink_1.cfg` : transient state on Γ_4 Figure 6.9: `heatsink_1.cfg` : transient state on Γ_1

3D cases

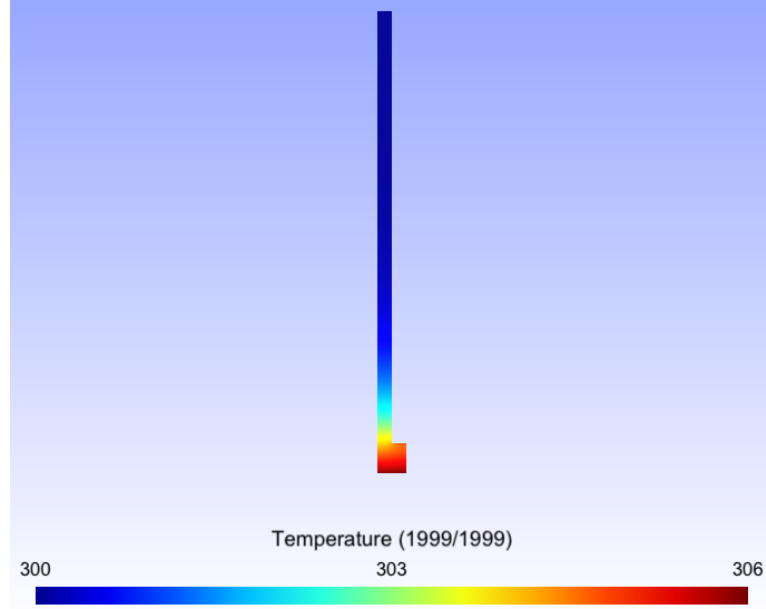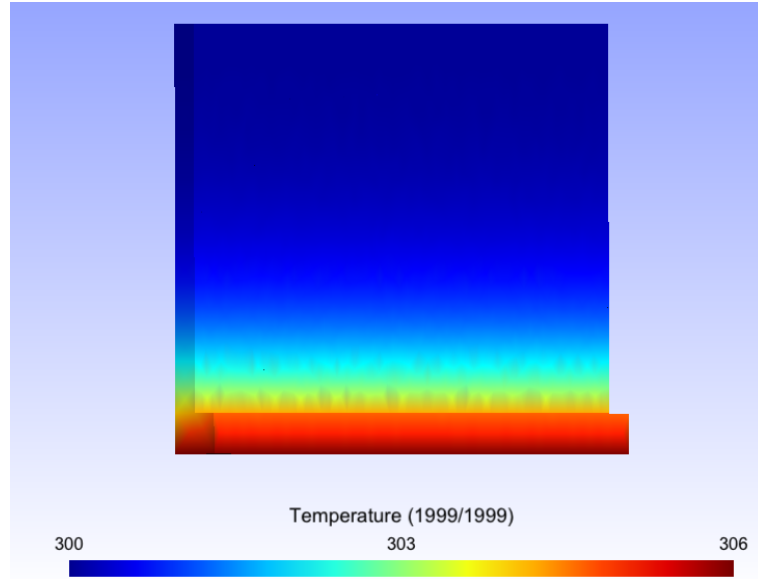Here is the result of 3D simulations considering the following configurations :

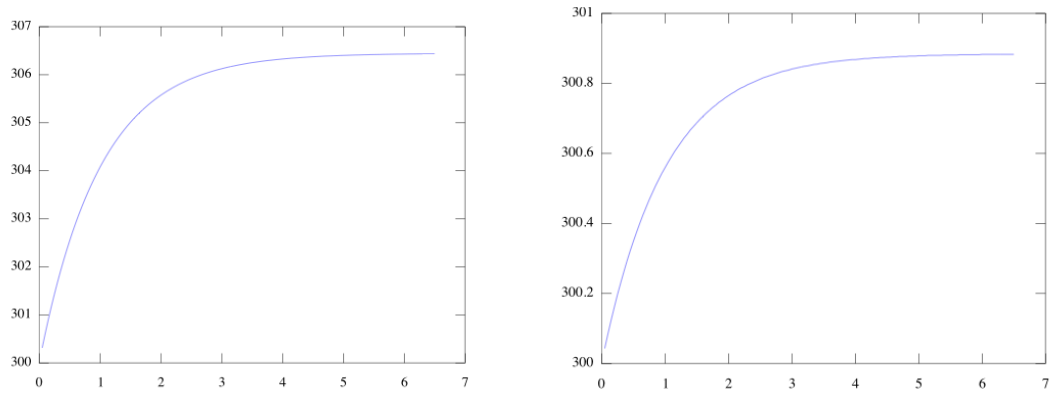Figure 6.10: `heatsink_3.cfg` : spreader and fin in copper, $Q = 1e6$ and $h = 1e3$

Figure 6.11: `heatsink_3.cfg` : transient state on Γ_4

Figure 6.12: `heatsink_3.cfg` : transient state on Γ_1

Christophe Prud'homme

CHAPTER 7

Natural convection in a heated tank

By Christophe Prud'homme

Chapter ref: **[cha:natural-convection-2d]**

7.1 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 domains. 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.

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 7.1. 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:

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

$$(7.1)$$

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.
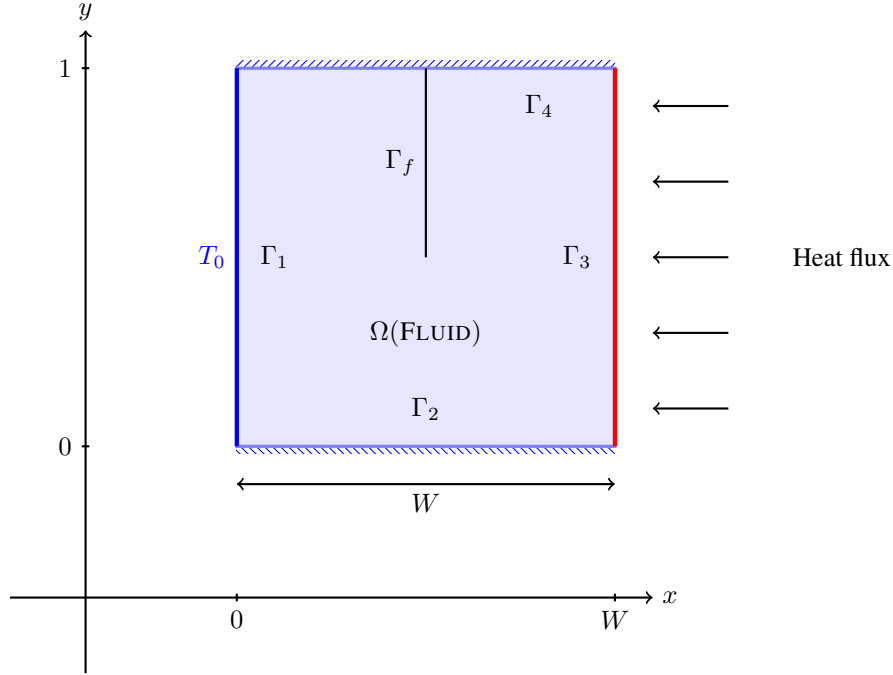
Figure 7.1: Geometry of the model

The heat equation reads:

$$\mathbf{u} \cdot \nabla\theta - \frac{1}{\sqrt{\mathrm{GrPr}}}\Delta\theta = 0$$

$$\theta = 0 \text{ sur } \Gamma_1$$

$$\frac{\partial\theta}{\partial n} = 0 \text{ sur } \Gamma_{2,4}$$

$$\frac{\partial\theta}{\partial n} = 1 \text{ sur } \Gamma_3$$

(7.2)

where Pr is the Prandtl number.

7.2 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 7.2. 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 $[1, 1e7]$.

7.3 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.
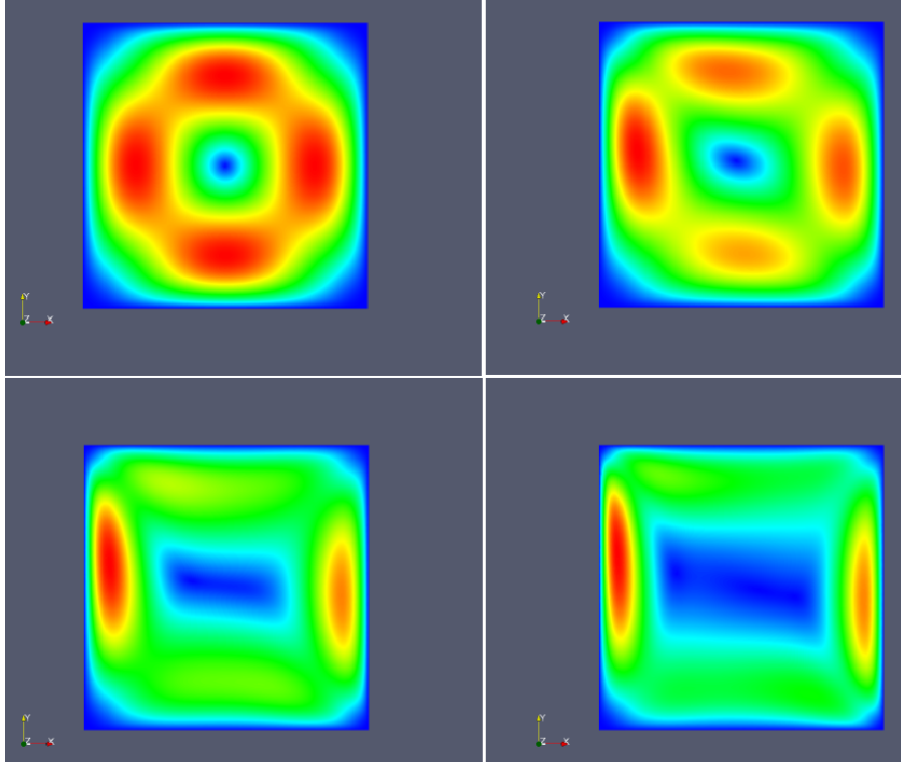
Figure 7.2: Velocity norm with respect to Grashof, $\mathrm{Gr} = 100, 10000, 100000, 500000$. $h = 0.01$ and $\mathrm{Pr} = 1$.

7.3.1 Mean temperature

We consider first the mean temperature on boundary Γ_3

$$T_3 = \int_{\Gamma_3} \theta \qquad (7.3)$$

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 7.3.

7.3.2 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 7.1. The flow rate, denoted D_f, reads

$$\mathrm{D}_f = \int_{\Gamma_f} \mathbf{u} \cdot \mathbf{e}_1 \qquad (7.4)$$

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.

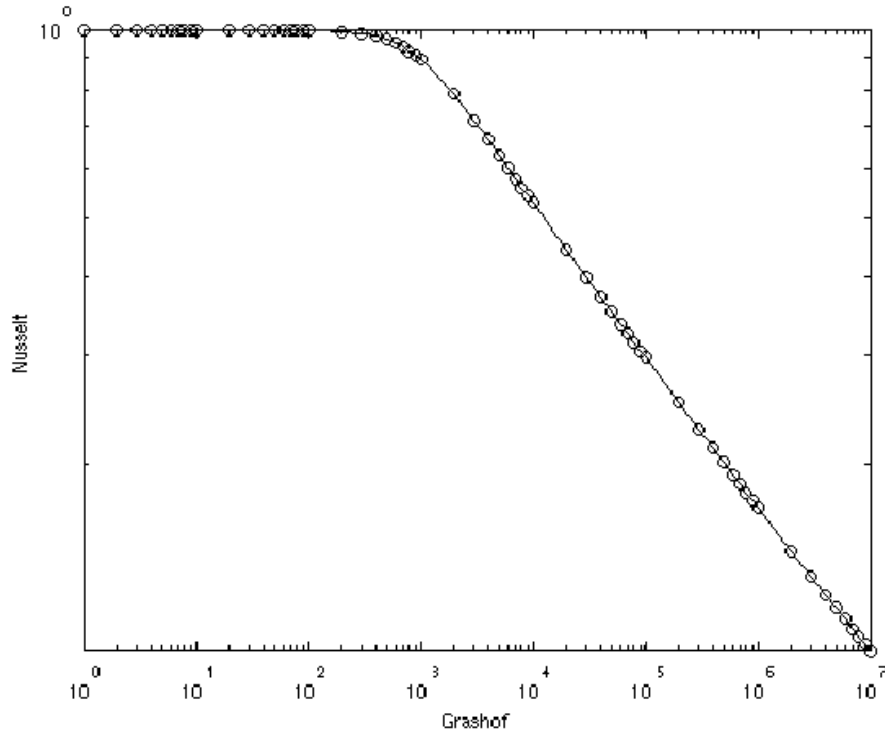Figure 7.3: Mean temperature with respect to the Grashof number; $h = 0.02$ with $\mathbb{P}_3$ Lagrange element for the velocity, $\mathbb{P}_2$ Lagrange for the pressure and $\mathbb{P}_1$ Lagrange for the temperature.
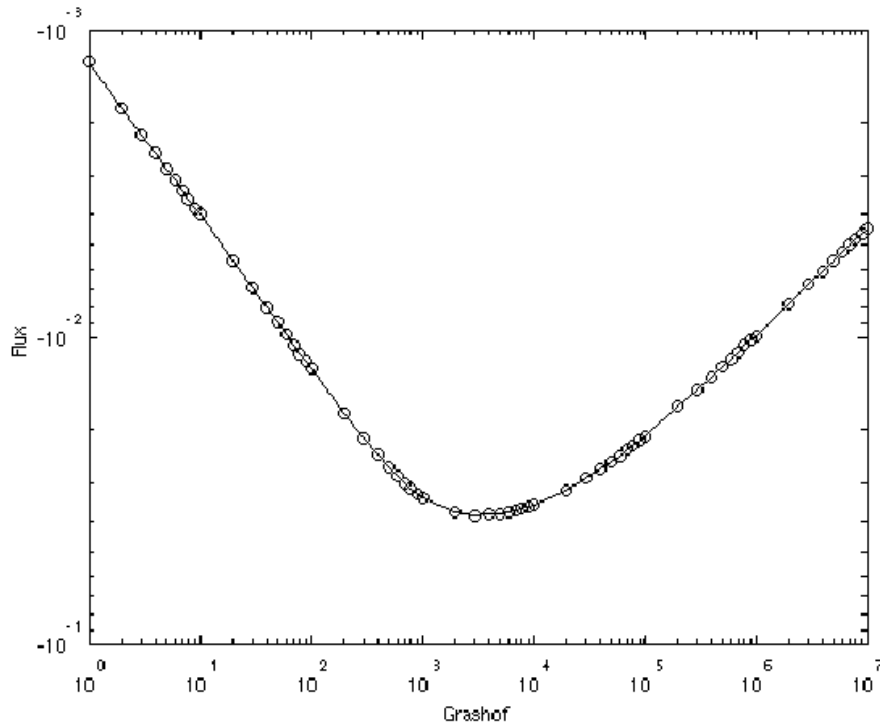

Figure 7.4: Behavior of the flow rate with respect to the Grashof number; $h = 0.02$, $\mathbb{P}_3$ for the velocity, $\mathbb{P}_2$ for the pressure and $\mathbb{P}_1$ for the temperature.

7.4 Implementation

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

7.5 Numerical Schemes

7.5.1 Stokes problem formulation and the pressure

7.5.2 The Stokes problem

Consider the following problem,

$$\text{Stokes:} \quad \begin{cases} -\mu \Delta \mathbf{u} + \nabla p = \mathbf{f} \\ \nabla \cdot \mathbf{u} = 0 \\ \mathbf{u}|_{\partial \Omega} = 0 \end{cases} \tag{7.5}$$

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 (7.5)

$$\int_\Omega p = 0 \tag{7.6}$$

Remark 2 (The Navier-Stokes case) *This is also true for the incompressible Navier-Stokes equations. We chose Stokes to simplify the exposure.*

7.5.3 Reformulation

In order to impose the condition (7.6), 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

$$\text{Stokes 2:} \quad \begin{cases} -\mu \Delta \mathbf{u} + \nabla p &=& \mathbf{f} \\ \nabla \cdot \mathbf{u} + \lambda &=& 0 \\ \mathbf{u}|_{\partial \Omega} &=& 0 \\ \int_\Omega p &=& 0 \end{cases} \tag{7.7}$$

Remark 3 (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 (7.6). 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.*

7.5.4 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 $(\mathbf{v}, q, \eta) \in \mathbf{H}_0^1(\Omega) \times L_0^2(\Omega) \times \mathbb{R}$

$$\text{Stokes 3:} \quad \begin{cases} \int_\Omega \left(\nabla \mathbf{u} : \nabla \mathbf{v} + \nabla \cdot \mathbf{v} p \right) &=& \int_\Omega \mathbf{f} \cdot \mathbf{v} \\ \int_\Omega \left(\nabla \cdot \mathbf{u} q + \lambda q \right) &=& 0 \\ \int_\Omega p \eta &=& 0 \end{cases} \tag{7.8}$$

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

$$\int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v} + \nabla\cdot\mathbf{v}p + \nabla\cdot\mathbf{u}q + \lambda q + \eta p = \int_\Omega \mathbf{f}\cdot\mathbf{v} \tag{7.9}$$

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 $L^2(\Omega)$.

7.5.5 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) );
// ...
```

7.5.6 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

$$\underbrace{\rho\mathbf{u}\cdot\nabla\mathbf{u}}_{\text{convection}} - \underbrace{\nu\Delta\mathbf{u}}_{\text{diffusion}} + \nabla p = \mathbf{f} \text{ on } \Omega$$
$$\nabla\cdot\mathbf{u} = 0 \tag{7.10}$$
$$\mathbf{u} = \mathbf{0} \text{ on } \partial\Omega$$

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 (7.10) is to use a fix point algorithm.

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

$$\rho\mathbf{u}^{(k-1)}\cdot\nabla\mathbf{u}^{(k)} - \nu\Delta\mathbf{u}^{(k)} + \nabla p^{(k)} = \mathbf{f} \text{ on } \Omega$$
$$\nabla\cdot\mathbf{u}^{(k)} = 0$$
$$\mathbf{u}^{(k)} = 0 \text{ on } \partial\Omega \tag{7.11}$$
$$(\mathbf{u}^{(0)}, p^{(0)}) = (\mathbf{0}, 0)$$

The system (7.11) 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 FEEL++:

```
// 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 = ||uᵏ − u⁽ᵏ⁻¹⁾|| + ||pᵏ − p⁽ᵏ⁻¹⁾||
} while( error > epsilon );
```

7.5.7 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

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

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

7.5.8 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

$$\text{Find } X \text{ such that } F(X) = 0 \tag{7.12}$$

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

$$J_F(X^{(n)})(X^{(n+1)} - X^{(n)}) = -F(X^{(n)}) \tag{7.13}$$

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

$$
\begin{aligned}
F_1((u_i)_i, (p_i)_i, (\theta_i)_i) &= \sum_{i,j} u_i u_j a(\phi_i, \phi_k, \phi_j) - \sum_i p_i b(\phi_k, \psi_i) + \sum_i \theta_i c(\rho_i, \phi_k) + \sum_i u_i d(\phi_i, \phi_k) &= 0 \\
F_2((u_i)_i, (p_i)_i, (\theta_i)_i) &= \sum_i u_i b(\phi_i, \psi_l) &= 0 \\
F_3((u_i)_i, (p_i)_i, (\theta_i)_i) &= \sum_{i,j} u_i \theta_j e(\phi_i, \rho_j, \rho_m) + \sum_i \theta_i f(\rho_i, \rho_m) - g(\rho_m) &= 0
\end{aligned}
\tag{7.14}
$$

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

$$
\begin{aligned}
a(\mathbf{u}, \mathbf{v}, \beta) &= \int_\Omega \mathbf{v}^T((\nabla \mathbf{u})\beta) \\
b(\mathbf{v}, p) &= \int_\Omega p(\nabla \cdot \mathbf{v}) - \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} p \\
c(\theta, \mathbf{v}) &= \int_\Omega \theta \mathbf{e}_2 \cdot \mathbf{v} \\
d(\mathbf{u}, \mathbf{v}) &= \frac{1}{\sqrt{\text{Gr}}}\left(\int_\Omega \nabla \mathbf{u} : (\nabla \mathbf{v})^T - \int_{\partial\Omega} ((\nabla \mathbf{u})\mathbf{n}) \cdot \mathbf{v} \right) \\
e(\mathbf{u}, \theta, \chi) &= \int_\Omega (\mathbf{u} \cdot \nabla \theta)\chi \\
f(\theta, \chi) &= \frac{1}{\sqrt{\text{GrPr}}}\left(\int_\Omega \nabla \theta \cdot \nabla \chi - \int_{\Gamma_1} (\nabla \theta \cdot \mathbf{n})\chi \right) \\
g(\chi) &= \frac{1}{\sqrt{\text{GrPr}}} \int_{\Gamma_3} \chi
\end{aligned}
\tag{7.15}
$$

Remark 5 *Note that the boundary integrals are kept in order to apply the weak Dirichlet boundary condition trick, see next section B.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

$$\frac{\partial F_1}{\partial u_i} = \sum_j u_j a(\phi_i, \phi_k, \phi_j) + \sum_i u_i a(\phi_i, \phi_k, \phi_j) + d(\phi_i, \phi_k) \tag{7.16}$$

- Deriving the first equation with respect to p_i we get

$$\frac{\partial F_1}{\partial p_i} = -b(\phi_k, \psi_l) \tag{7.17}$$

- Deriving the first equation with respect to θ_i we get

$$\frac{\partial F_1}{\partial \theta_i} = c(\rho_i, \rho_k) \tag{7.18}$$

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

$$\frac{\partial F_2}{\partial u_i} = b(\phi_i, \psi_l) \tag{7.19}$$

Finally the third component

- Deriving with respect to u_i

$$\frac{\partial F_3}{\partial u_i} = \sum_j \theta_j e(\phi_i, \rho_j, \rho_m) \tag{7.20}$$

- Deriving with respect to p_i,

$$\frac{\partial F_3}{\partial p_i} = 0 \tag{7.21}$$

- Deriving with respect to $theta_i$,

$$\frac{\partial F_3}{\partial \theta_i} = \sum_j u_j e(\phi_j, \rho_i, \rho_m) + f(\rho_i, \rho_m) \tag{7.22}$$

$$J_F = \begin{pmatrix} \frac{\partial F_1}{\partial u_i} & \frac{\partial F_1}{\partial p_i} & \frac{\partial F_1}{\partial \theta_i} \\ \frac{\partial F_2}{\partial u_i} & \frac{\partial F_2}{\partial p_i}(=0) & \frac{\partial F_2}{\partial \theta_i}(=0) \\ \frac{\partial F_3}{\partial u_i} & \frac{\partial F_3}{\partial p_i}(=0) & \frac{\partial F_3}{\partial \theta_i} \end{pmatrix} \tag{7.23}$$

In order to implement J_F and solve (7.13), J_F can be expressed as the matrix associated with the discretisation of

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}, \beta_1) + a(\beta_1, \mathbf{v}, \mathbf{u}) + d(\mathbf{u}, \mathbf{v}) - b(\mathbf{v}, p) + c(\theta, \mathbf{v}) &= 0 \\ b(\mathbf{u}, q) &= 0 \\ e(\beta_1, \theta, \chi) + f(\theta, \chi) + e(\mathbf{u}, \beta_2, \chi) &= 0 \end{aligned} \tag{7.24}$$

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

FEEL++ Implementation

Now we use the FEEL++ non linear framework in order to implement our Newton scheme (7.13). 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 6 *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.

```cpp
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.

Thomas Strub, Philippe Helluy, Christophe Prud'homme

CHAPTER 8

2D Maxwell simulation in a diode

By Thomas Strub, Philippe Helluy, Christophe Prud'homme

Chapter ref: **[cha:maxwell-2d]**

8.1 Description

The Maxwell equations read:

$$\begin{aligned} \frac{-1}{c^2}\frac{\partial \boldsymbol{E}}{\partial t} + \nabla \times \boldsymbol{B} &= \mu_0 \boldsymbol{J} \\ \boldsymbol{B}_t + \nabla \times \boldsymbol{E} &= 0 \\ \nabla \cdot \boldsymbol{B} &= 0 \\ \nabla \cdot \boldsymbol{E} &= \frac{\rho}{\epsilon_o} \end{aligned}$$

where $\boldsymbol{E}$ is the electric field, $\boldsymbol{B}$ the magnetic field, $\boldsymbol{J}$ the current density, c the speed of light, rho density of electric charge, mu_0 the vacuum permeability and $epsilon_0$ the vacuum permittivity.

In the midst industrial notament in aeronautics, systems Products must verify certain standards such as the receipt an electromagnetic wave emitted by a radar does not cause the inefficassité of part or all of the hardware in the system.

Thus, the simulation of such situations can develop when or during the certification of a new product to test its reaction to such attacks.

Also note that the last two equations are actually initial conditions, since if we assume they are true at the moment $t = 0$ then it can be deduced from the first two.

At $t = 0s$, we suppose that

$$\nabla \cdot \boldsymbol{B} = 0 \tag{8.1}$$

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon_o} \tag{8.2}$$

Suppose that $\boldsymbol{B} = (B_x, B_y, B_z)^T$ and $\boldsymbol{E} = (E_x, E_y, E_z)^T$ i.e.

$$\frac{\partial B_x}{\partial x}(t=0) + \frac{\partial B_y}{\partial y}(t=0) + \frac{\partial B_z}{\partial z} \quad (t=0) = 0 \tag{8.3}$$

$$\frac{\partial E_x}{\partial x}(t=0) + \frac{\partial E_y}{\partial y}(t=0) + \frac{\partial E_z}{\partial z} \quad (t=0) = \frac{\rho}{\epsilon_o} \tag{8.4}$$

Differentiating the first of these two equations with respect to time, we get:

$$\frac{\partial}{\partial t}\frac{\partial}{\partial x}B_x + \frac{\partial}{\partial t}\frac{\partial}{\partial y}B_y + \frac{\partial}{\partial t}\frac{\partial}{\partial z}B_z =$$

$$\frac{\partial}{\partial x}\left(\frac{\partial}{\partial y}E_z - \frac{\partial}{\partial z}E_y\right) + \frac{\partial}{\partial y}\left(\frac{\partial}{\partial z}E_x - \frac{\partial}{\partial x}E_z\right) + \frac{\partial}{\partial z}\left(\frac{\partial}{\partial x}E_y - \frac{\partial}{\partial y}E_x\right)$$

$$= 0 \quad (8.5)$$

thanks to

$$\boldsymbol{B}_t + \nabla \times \boldsymbol{E} = 0 \tag{8.6}$$

So, for all $t \geq 0$,

$$\nabla \cdot \boldsymbol{B}(t) = \nabla \cdot \boldsymbol{B}(0) = 0 \tag{8.7}$$

We deduce the same way the second equation, using the charge conservation equation :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{J}) = 0 \tag{8.8}$$

8.2 Variational formulation

8.3 Implementation

8.4 Numerical Results

Abdoulaye Samake, Vincent Chabannes, Christophe Prud'homme

CHAPTER 9

Domain decomposition methods

By Abdoulaye Samake, Vincent Chabannes, Christophe Prud'homme

---

Chapter ref: **[cha:dd]**

9.1 A Really Short Introduction

In mathematics, numerical analysis, and numerical partial differential equations, domain decomposition methods solve a boundary value problem by splitting it into smaller boundary value problems on subdomains and iterating to coordinate the solution between the adjacent subdomains. A corse problem with one or fiew unknowns per subdomain is used to further coordinate the solution between the subdomains globally.

9.2 A 1D model

We consider the following laplacian boundary value problem

$$\begin{cases} -u"(x) = f(x) \quad \text{in} \quad]0,1[\\ u(0) = \alpha, \ u(1) = \beta \end{cases} \tag{9.1}$$

where $\alpha, \beta \in \mathbb{R}$.

9.2.1 Schwartz algorithms

The schwartz overlapping multiplicative algorithm with dirichlet interface conditions for this problem at n^{th} iteration is given by

$$\begin{cases} -u_1"^n(x) = f(x) \quad \text{in} \quad]0,b[\\ \quad u_1^n(0) = \alpha \\ \quad u_1^n(b) = u_2^{n-1}(b) \end{cases} \quad \text{and} \quad \begin{cases} -u_2"^n(x) = f(x) \quad \text{in} \quad]a,1[\\ \quad u_2^n(1) = \beta \\ \quad u_2^n(a) = u_1^n(a) \end{cases} \tag{9.2}$$

where $n \in \mathbb{N}^*, a, b \in \mathbb{R}$ and $a < b$.
Let $e_i^n = u_i^n - u \ (i = 1,2)$, the error at n^{th} iteration relative to the exact solution, the convergence rate is given by

$$\rho = \frac{|e_1^n|}{|e_1^{n-1}|} = \frac{a}{b}\frac{1-b}{1-a} = \frac{|e_2^n|}{|e_2^{n-1}|}. \tag{9.3}$$

71

9.2.2 Variational formulations

find u such that

$$\int_0^b u_1' v' = \int_0^b fv \quad \forall v \qquad \text{in the first subdomain } \Omega_1 =]0, b[$$

$$\int_a^1 u_2' v' = \int_a^1 fv \quad \forall v \qquad \text{in the second subdomain } \Omega_2 =]a, 1[$$

9.3 A 2 domain overlapping Schwartz method in 2D and 3D

We consider the following laplacian boundary value problem

$$\begin{cases} -\Delta u = f & \text{in} \quad \Omega \\ u = g & \text{on} \quad \partial\Omega \end{cases} \tag{9.4}$$

where $\Omega \subset \mathbb{R}^d, d = 2, 3$ and g is the dirichlet boundary value.

9.3.1 Schwartz algorithms

The schwartz overlapping multiplicative algorithm with dirichlet interface conditions for this problem on two subdomains Ω_1 and Ω_2 at n^{th} iteration is given by

$$\begin{cases} -\Delta u_1^n = f & \text{in} \quad \Omega_1 \\ u_1^n = g & \text{on} \quad \partial\Omega_1^{ext} \\ u_1^n = u_2^{n-1} & \text{on} \quad \Gamma_1 \end{cases} \quad \text{and} \quad \begin{cases} -\Delta u_2^n = f & \text{in} \quad \Omega_2 \\ u_2^n = g & \text{on} \quad \partial\Omega_2^{ext} \\ u_2^n = u_1^n & \text{on} \quad \Gamma_2 \end{cases} \tag{9.5}$$

9.3.2 Variational formulations

$$\int_{\Omega_i} \nabla u_i \cdot \nabla v = \int_{\Omega_i} fv \quad \forall\, v,\, i = 1, 2.$$

FEEL++ implementation

```
/*
  Implementation of the local problem
*/
template<Expr>
void
localProblem(element_type& u, Expr expr)
{
// Assembly of the right hand side ∫_Ω fv
    auto F = M_backend->newVector(Xh);
    form1( _test=Xh,_vector=F, _init=true ) =
        integrate( elements(mesh), f*id(v) );
    F->close();

// Assembly of the left hand side ∫_Ω ∇u·∇v
    auto A = M_backend->newMatrix( Xh, Xh );
    form2( _test=Xh, _trial=Xh, _matrix=A, _init=true ) =
        integrate( elements(mesh), gradt(u)*trans(grad(v)) );
    A->close();

// Apply the dirichlet boundary conditions
    form2( Xh, Xh, A ) +=
        on( markedfaces(mesh, "Dirichlet") ,u,F,g);

// Apply the dirichlet interface conditions
```

```
      form2( Xh, Xh, A ) +=
             on( markedfaces(mesh, "Interface") ,u,F,expr);

 // solve the linear system Au = F
    M_backend->solve(_matrix=A, _solution=u, _rhs=F );
}

  unsigned int cpt = 0;
  double tolerance = 1e − 8;
  double maxIterations = 20;
  double l2erroru₁ = 1.;
  double l2erroru₂ = 1;
  /*
  Iteration loop
  */
  while( (l2erroru₁ +l2erroru₂) > tolerance && cpt <= maxIterations)
  {

     // call the localProblem on the first subdomain Ω₁
        localProblem(u₁, idv(u₂));

     // call the localProblem on the first subdomain Ω₂
        localProblem(u₂, idv(u₁));

     // compute L2 errors on each subdomain
        L2erroru₁ = l2Error(u₁);
        L2erroru₂ = l2Error(u₂);

     // increment the cunter
        ++cpt;
  }
```

9.3.3 Numerical results in 2D case

The numerical results presented in the following table correspond to the partition of the global domain Ω in two subdomains Ω_1 and Ω_2 (see figure 9.2) and the following configuration:

1. $g(x, y) = \sin(\pi x) \cos(\pi y)$: the exact solution

2. $f(x, y) = 2\pi^2 g$: the right hand side of the equation

3. $\mathbb{P}_2$ approximation : the lagrange polynomial order

4. hsize $= 0.02$: the mesh size
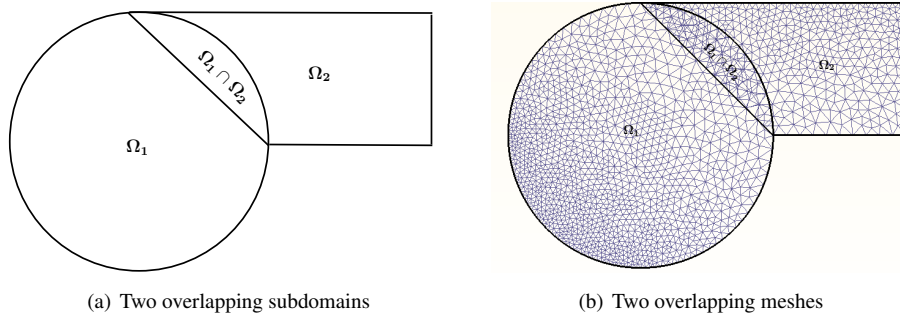
5. tol $= 1e - 9$: the tolerance

(a) Two overlapping subdomains   (b) Two overlapping meshes

Figure 9.1: geometry

Number of iterations	$\|\mathbf{u_1} - \mathbf{u_{ex}}\|_{\mathbf{L_2}}$	$\|\mathbf{u_2} - \mathbf{u_{ex}}\|_{\mathbf{L_2}}$
11	2.52e-8	2.16e-8

9.3.4 Numerical solutions in 2D case

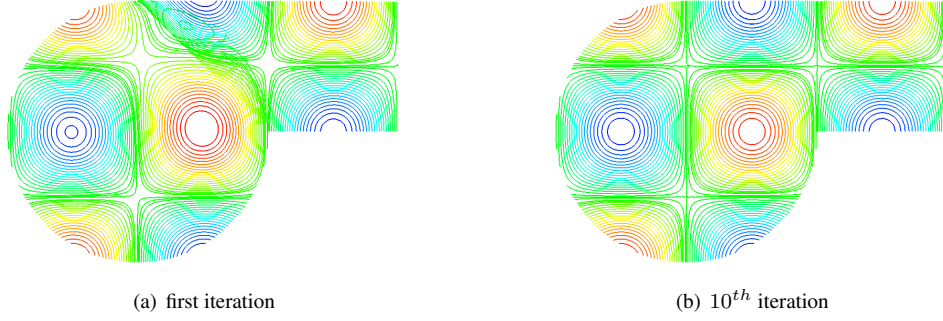(a) first iteration   (b) 10th iteration

Figure 9.2: isovalues of solution in 2D

9.4 Computing the eigenmodes of the Dirichlet to Neumann operator

9.4.1 Problem description and variational formulation

We consider at the continuous level the Dirichlet-to-Neumann(DtN) map on Ω, denoted by DtN$_\Omega$.

Let $u : \Gamma \longmapsto \mathbb{R}$,

$$\text{DtN}_\Omega(u) = \kappa \frac{\partial v}{n}\Big|_\Gamma$$

where v satisfies

$$\begin{cases} \mathcal{L}(v) := (\eta - \text{div}(\kappa\nabla))v = 0 & \text{dans} \quad \Omega, \\ v = u & \text{sur} \quad \Gamma \end{cases} \tag{9.6}$$

where Ω is a bounded domain of $\mathbb{R}^d$ (d=2 or 3), and Γ it border, κ is a positive diffusion function which can be discontinuous, and $\eta \geq 0$. The eigenmodes of the Dirichlet-to-Neumann operator are solutions of the following eigenvalues problem

$$\text{DtN}_\Omega(u) = \lambda\kappa u \tag{9.7}$$

To obtain the discrete form of the DtN map, we consider the variational form of (9.6). let's define the bilinear form $a : H^1(\Omega) \times H^1(\Omega) \longrightarrow \mathbb{R}$,

$$a(w, v) := \int_\Omega \eta wv + \kappa\nabla w \cdot \nabla v.$$

With a finite element basis $\{\phi_k\}$, the coefficient matrix of a Neumann boundary value problem in Ω is

$$A_{kl} := \int_\Omega \eta\phi_k\phi_l + \kappa\nabla\phi_k \cdot \nabla\phi_l.$$

A variational formulation of the flux reads

$$\int_\Gamma \kappa\frac{\partial v}{\partial n}\phi_k = \int_\Omega \eta v\phi_k + \kappa\nabla v \cdot \nabla\phi_k \quad \forall \phi_k.$$

74

So the variational formulation of the eigenvalue problem (9.7) reads

$$\int_\Omega \eta v \phi_k + \kappa \nabla v \cdot \nabla \phi_k = \lambda \int_\Gamma \kappa v \phi_k \quad \forall \, \phi_k. \tag{9.8}$$

Let B be the weighted mass matrix

$$(B)_{kl} = \int_\Gamma \kappa \phi_k \phi_l$$

The compact form of (9.8) is

$$Av = \lambda Bv \tag{9.9}$$

FEEL++ implementation

```cpp
// Assembly of the right hand side B = ∫_Γ κvw
 auto B = M_backend->newMatrix( Xh, Xh ) ;
 form2( _test=Xh, _trial=Xh, _matrix=B, _init=true );
 BOOST_FOREACH( int marker, flags )
 {
    form2( Xh, Xh, B ) +=
    integrate( markedfaces(mesh,marker), kappa*idt(u)*id(v) );
 }
 B->close();
// Assembly of the left hand side A = ∫_Ω ηvw + κ∇v · ∇w
 auto A = M_backend->newMatrix( Xh, Xh ) ;
 form2( _test=Xh, _trial=Xh, _matrix=A, _init=true ) =
 integrate( elements(mesh), kappa*gradt(u)*trans(grad(v)) + nu*idt(u)*id(v) );
 A->close();

// eigenvalue solver options
 int nev = this->vm()["solvereigen-nev"].template as<int>();
 int ncv = this->vm()["solvereigen-ncv"].template as<int>();;
// definition of the eigenmodes
 SolverEigen<double>::eigenmodes_type modes;
// solve the eigenvalue problem Av = λBv
 modes=
      eigs( _matrixA=A,
            _matrixB=B,
            _nev=nev,
            _ncv=ncv,
            _transform=SINVERT,
            _spectrum=SMALLEST_MAGNITUDE,
            _verbose = true );

}
```

9.4.2 Numerical solutions

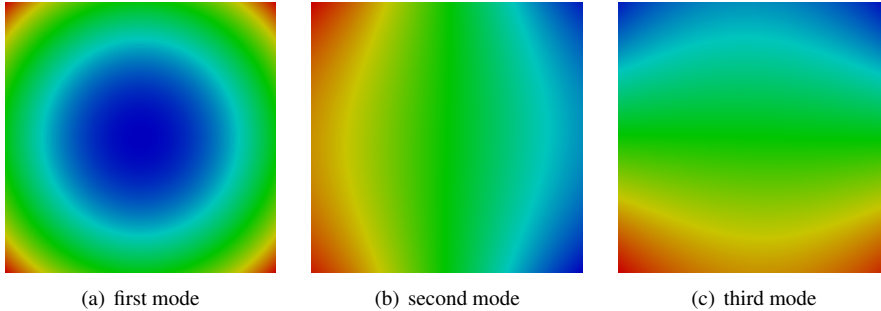(a) first mode   (b) second mode   (c) third mode

Figure 9.3: three eigenmodes

These numerical solutions correspond to the following configuration :

1. $\mathbb{P}_2$ approximation : the lagrange polynomial order

2. hsize $= 0.02$: the mesh size

3. $\mu = \kappa = 1$.

Abdoulaye Samake, Vincent Chabannes, Christophe Prud'homme

Part III

Programming with FEEL++

Part IV

Appendix

APPENDIX A

How to ?

A.1 Introduction

This section includes the FAQ avaible on Feel web site, if you want to post a question, please visit it and follow the instruction to edit the FAQ.

A.2 Meshes

A.2.1 *What are the main execution options of a* FEEL++ *application ?*

Let's consider that your application is named feelapp, in that case you can modify the main execution options of your application with

▐ ./feelapp --shape="simplex" --nochdir --exporter-format=gmsh

These options are :

- shape=["simplex","hypercube"] which is the shape of the generated mesh

- nochdir means that you want the result in the current directory (by default in ~/feel)

- exporter-format enables you to choose the format of mesh results output

A.2.2 *How to create a mesh?*

Here is an example of how to create a mesh with GMSH generator :

```
mesh_ptrtype mesh =
      createGMSHMesh( _mesh=new mesh_type,
      _update=MESH_CHECK!MESH_UPDATE_FACES!MESH_UPDATE_EDGES!MESH_RENUMBER,
      _desc=domain( _name= (boost::format( "%1%-%2%-%3%" ) %"hypercube" %Dim %1).str(),
      _shape="hypercube",
      _dim=Dim,
      _h=meshSize,
      _xmin=-1.,
      _xmax=1.,
      _ymin=-1.,
      _ymax=1. ) );
```

Here is an example of how to create a mesh with a .geo file :

```
mesh_ptrtype mesh =
        createGMSHMesh( _mesh=new mesh_type,
         _update=MESH_CHECK!MESH_UPDATE_FACES!MESH_UPDATE_EDGES!MESH_RENUMBER,
         _desc="???" );
```

A.2.3 *What are the different parameters of the function domain() ?*

The function `domain()` is located in `feel/feel/feefilters/gmsh.hpp` and enables to generate a simple geometrical domain from required and optional parameters. Its avaible options are :

- `_name = "string"` gives the prefix of the gmsh geo and mesh files,

- `_shape = "simplex"`, `"hypercube"`, `"ellipsoid"` gives the shape of the domain, it is one of these three possibilities

- `_dim = 1, 2` **or** `3` gives the topological dimension of the domain. For example if `_dim=2` and `_shape="simplex"` this will produce a triangle

- `_h = real value` gives the characteristic size of the mesh, e.g. `_h=0.1`

- `_xmin = real` gives the minimum x value of the domain for example `_xmin=-1`

- `_xmax = real` gives the maximum x value of the domain for example `_xmax=-1`

- `_ymin = real` gives the minimum x value of the domain for example `_ymin=-1`.

- `_ymax = real` gives the maximum y value of the domain for example `_ymax=1`.

A.2.4 *How to loop on the degrees of freedom coordinates of a function ?*

Take a look at the example which is in `feel/examples/snippets/dofpoints.cpp`

A.2.5 *How to work with specific meshes ?*

`loadmesh.cpp`

FEEL++ supports several meshes file formats. It supports essentially Gmsh mesh file format but other are acceptable, with some modifications :

- medit (`.mesh`)
  There is a small difference between medit meshes and gmsh ones. The medit reader of Gmsh is able to read medit meshes, the issue comes from markers for areas of the edges were we want to apply different boundary conditions. Gmsh is currently using the Physical Entities (physical line, area, volume). Unfortunetly, the medit reader of Gmsh considers the physical flag as null (to go deeper, you can check this part on Gmsh web site). This option is took into account in FEEL++, the only modification is to put the optional parameter `physical_are_elementary_regions` as **true** in both functions `createGMSHMesh` and/or `loadGMSHMesh`. We have prepared a simple example which imports a `medit` mesh with a surface and volume calculation on it. You can find it in `feel/doc/manual/loadmesh.cpp`.

  Please not that furthers `medit` meshes are presented in example in the directory `/feel/data/medit/`. The `geo` scripts are those which are produced by FEEL++ when reading those meshes.

- Stl (`.stl`)
  You can also use `stl` files, those files are native to the stereolithography CAD software created by 3D Systems. These files describe only the surface geometry of a three dimensional object without any representation of color, texture or other common attributes. You have further examples of such files in `feel/data/stl`.

  To use FEEL++ with `stl` files, you have to create a `geo` script to enable gmsh to remesh the file. The `stl` file you want to use has to be a volume mesh. The script is very small, you have all informations to make one at Gmsh/slt section on their web site. Once it's done, you juste have to type

> gmsh stl_file_name.geo -3

with `stl_file_name.stl` in the same directory. That command will produce you the correct `.msh` mesh that you could now use as usual without any modification in your FEEL++ application.

Take a look above how the remesh has produced a complete mesh with the file `pelvis.stl` and `pelvis.geo`:
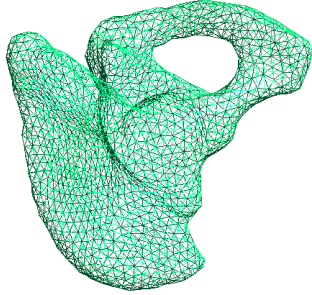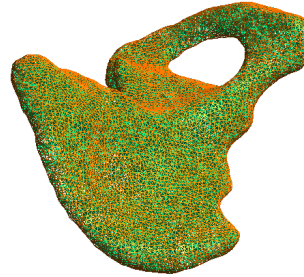

Figure A.1: Pelvis before remesh (stl)

Figure A.2: Pelvis after remesh (msh)

A.3 Language for Partial Differential Equations

A.3.1 What is the difference between using the "vf::project" function and solve a weak projection problem ?

To make it clear, let's considerate that we want to project a $\mathbb{P}_1$ scalar function σ on a $\mathbb{P}_0$ space. We have two alternatives to do it :

- Computing the $\mathcal{L}_2$ projection of σ onto the space
  Here $kappa$ and v are $\mathbb{P}_0$ functions :
  ```
  Matrix_M=integrate(elements(mesh), idt(kappa) * id(v));
  Vector_F=integrate(elements(mesh), idv(sigma) * id(v));
  ```

- Use the project function `vf::project`
  This function does a nodal projection : at the dof point the projection will be <u>exactly</u> equal to the projected function σ. It works as follow
  ```
  kappa=vf::project(P0_space, elements(mesh), idv(sigma));
  ```

These two projections are in general different, if you compare the values in the vector, they will be (slightly) different. However as $h \rightarrow 0$ they should both converge to the σ function.

A.3.2 How to do a quick L2 projection of an expression ?

Let say that we have created two spaces, one scalar and one vectorial, we call them X_h and X_{hVec} and one wants to project some expressions on those spaces.

For example, we want to project $(x,y) \rightarrow \sqrt{x^2 - y^2} - 1$ on the scalar space and $(-2y, \cos x)$ on the vectorial space. First of all, one has to create projectors for the scalar and vectorial spaces, the code reads as follow :

```
#include <feel/feeldiscr/projector.hpp>
auto l2p = projector(Xh, Xh);
auto l2pVec = projector(XhVec, XhVec);
```

You can note that `projector(Space, Space)` returns a `boost::shared_ptr` on a `Projector` object which makes projecting functions on `Space` possible.

Then, one uses the function `Projector::project(Expression)` :

```
auto Circle = l2p->project( sqrt( pow((vf::Px()),2.0)+ pow((vf::Py()),2.0)) - 1 );

auto F = l2pVec->project( -2 * Py() * oneX() + cos(vf::Px()) * oneY() );
```

Here you can note that the types of `Circle` and `F` are respectively : $X_h\_type :: element\_type$ and $X_{hVec}\_type :: element\_type$
An equivalent way to write it is to use the `Projector::operator()(Expression)` :

```
auto Circle = (*l2p)( sqrt( pow((vf::Px()),2.0)+ pow((vf::Py()),2.0)) - 1 );

auto F = (*l2pVec)( -2 * Py() * oneX() + cos(vf::Px()) * oneY() );
```

`Projector::operator()` accepts many types of arguments, see `feel/feeldiscr/projector.hpp` for details.

A.3.3   How to compose FEEL++ operators ?

Let's considerate that we have created two spaces, one scalar X_h and one vectorial X_{hVec}. We also have two vectors a and b (of type $X_h\_type :: element\_type$).

One wants to do the following operation : $div(grad(a*b))$. The following expression is **not** yet implemented in FEEL++:

```
divv( gradv( idv(a) * idv(b) ) )
```

One has to do intermediate projections to compose the operators. Using the Projector class, the code reads :

```
#include <feel/feeldiscr/projector.hpp>

// create projectors on Xh and XhVec spaces
auto l2p = projector(Xh, Xh);
auto l2pVec = projector(XhVec, XhVec);

auto ab = l2p->project( idv(a)*idv(b) );
auto grad_ab = l2pVec->project( gradv(ab) );
auto div_grad_ab = l2p->project( divv(grad_ab) );
```

Here `div_grad_ab` has the type $X_h\_type :: element_type$. There is an equivalent but verboseless way to write this composition : use the `Projector::operator()` which accepts has argument an expression or an `element_type`. So one could write :

```
#include <feel/feeldiscr/projector.hpp>

//create projectors on Xh and XhVec spaces
auto l2p = projector(Xh, Xh);
auto l2pVec = projector(XhVec, XhVec);

auto div_grad_ab = (*l2p)( divv( (*l2pVec)( gradv( (*l2p)(idv(a)*idv(b)) ) ) ) );
```

the * is needed before `l2p` or `l2pVec` since there are `boost::shared_ptr` objects. One could also create directly `Projector` objects :

```
Projector<Xh_type, Xh_type> l2p(Xh, Xh);

auto ab = l2p(idv(a)*idv(b));
```

APPENDIX B

Random notes

B.1 Becoming a Feel++ developer

B.1.1 *Interest*

Becoming a FEEL++ developer makes library improvements possible, you may have several proposals which may be usefull. Taking part of the project will enable you to commit some modifications or new applications, we will be glad to count you among us. As an open-source project under GNU licence, you will be able to commit and participate to the entire project and its various aspects. Our aim is that each user should be involved in the library's expansion. In the following part, you will see how you can become a FEEL++ collaborator.

B.1.2 *Creating RSA keys*

At the top of the manual, you have seen how to get the sources anonymously, if you want to checkout or commit properly, you will need an account on . After the administrator approval, you have to demand the rights to see the project tree.

Once it's done, 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` and accept the 3 questions without typing anything. The generated key is placed in `~/.ssh/id_rsa.pub`, you just need to copy this file's content in your forge account. To make it, go on the Forge website and enter into your account's personnal page. At the bottom of the page, you'll have the possibility to edit your SSH keys, go into it and copy/paste the id_rsa.pub content. Once it's done, the number of your SSH keys in that page should have increased. Now, 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 Forge, you must add the commands in the `~/.ssh/config` file :

```
host forge.imag.fr
  user <your_login_forge>
```

B.1.3 *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@scm.forge.imag.fr/var/lib/gforge/chroot/scmrepos
/svn/life/trunk/life/trunk feel
```
where `login` is your login name in the Forge plateform.

You are now able to checkout, commit or add the file your judge usefull using `svn`, please don't forget to comment on your various actions. The first commit is subject to the approbation of one of the main developers.

B.2   Linear Algebra with PETSC

B.2.1   *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).

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

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

B.2.2   *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

B.2.3   *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

B.3 Weak Dirichlet boudary conditions

B.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 7 *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 8 *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 9 *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 exactly but the convergence order remain optimal*

Remark 10 *Weak treatment: Disadvantages*

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

Advantages, disadvantages

Remark 11 *Strong treatment: Advantages*

- *Enforce exactly the boundary conditions*

Remark 12 *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.*

B.3.2 *Laplacian*

Example: Laplacian

$$- \Delta u = f \,(\text{non conservative}), \quad -\nabla \cdot (\nabla u) = f \,(\text{conservative}), \quad u = g|_{\partial\Omega} \qquad (B.2)$$

the flux is vector ∇u

$$\int_\Omega \nabla u \cdot \nabla v + \int_{\partial\Omega} \underbrace{-\frac{\partial u}{\partial n} v}_{\text{integration by part}} \quad \underbrace{-\frac{\partial v}{\partial n} u}_{\text{adjoint consistency: symetrisation}} \quad + \quad \underbrace{\frac{\gamma}{h} uv}_{\text{penalisation: enforce Dirichlet condition}} \qquad (B.3)$$

$$\int_\Omega f \nabla v + \int_{\partial\Omega} (\quad \underbrace{-\frac{\partial v}{\partial n} g}_{\text{adjoint consistency}} \quad + \quad \underbrace{\frac{\gamma}{h} v) g}_{\text{penalisation: enforce Dirichlet condition}} \qquad (B.4)$$

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
```

B.3.3 *Convection-Diffusion*

Example: Convection-Diffusion

Remark 13 *Convection Diffusion Consider now the following problem, find u such that*

$$- \Delta u + \mathbf{c} \cdot \nabla u = f, \quad u = g|_{\partial\Omega}, \quad \nabla \cdot \mathbf{c} = 0 \qquad (B.5)$$

under conservative form the equation reads

$$\nabla \cdot (-\nabla u + \mathbf{c} u) = f, \quad u = g|_{\partial\Omega}, \quad \nabla \cdot \mathbf{c} = 0 \qquad (B.6)$$

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

$$\nabla \cdot (\mathbf{c} u) = \mathbf{c} \cdot \nabla u + \underbrace{u \nabla \cdot \mathbf{c}}_{=0} \qquad (B.7)$$

Weak formulation for convection diffusion

Multiplying by any test function v and integration by part of (B.6) gives

$$\int_\Omega \nabla u \cdot \nabla v + (\mathbf{c} \cdot \nabla u) v + \int_{\partial\Omega} (\mathbf{F} \cdot \mathbf{n}) v = \int_\Omega f v \qquad (B.8)$$

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 $\partial\Omega$. (B.8) reads now

$$\int_\Omega \nabla u \cdot \nabla v + (\mathbf{c} \cdot \nabla u) v + \int_{\partial\Omega} (\mathbf{F} \cdot \mathbf{n}) v + \frac{\gamma}{\mathbf{h}} \mathbf{u} \mathbf{v} = \int_\Omega f v + \int_{\partial\Omega} \frac{\gamma}{\mathbf{h}} \mathbf{g} \mathbf{v} \qquad (B.9)$$

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

$$\int_\Omega \nabla u \cdot \nabla v + (\mathbf{c} \cdot \nabla u)v + \int_{\partial\Omega} ((-\nabla u + \mathbf{c}u) \cdot \mathbf{n})v + ((-\nabla v + \mathbf{c}v) \cdot \mathbf{n})\mathbf{u} + \frac{\gamma}{h}uv = $$
$$\int_\Omega fv + \int_{\partial\Omega} ((-\nabla v + \mathbf{c}v) \cdot \mathbf{n})\mathbf{g} + \frac{\gamma}{h}gv \tag{B.10}$$

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());
```

B.3.4   *Stokes*

Example: Stokes

Remark 14 *Stokes Consider now the following problem, find* $(\mathbf{u}, p)$ *such that*

$$-\Delta\mathbf{u} + \nabla p = \mathbf{f}, \quad \mathbf{u} = \mathbf{g}|_{\partial\Omega}, \quad \nabla \cdot \mathbf{u} = 0 \tag{B.11}$$

under conservative form the equation reads

$$\nabla \cdot (-\nabla\mathbf{u} + p\mathbb{I}) \quad = \mathbf{f}, \tag{B.12}$$
$$\nabla \cdot \mathbf{u} \quad = 0, \tag{B.13}$$
$$\mathbf{u} \quad = \mathbf{g}|_{\partial\Omega} \tag{B.14}$$

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*

$$\nabla \cdot (\mathbb{M}\mathbf{v}) = (\nabla \cdot \mathbb{M}) \cdot \mathbf{v} + \mathbb{M} : (\nabla\mathbf{v}) \tag{B.15}$$

where $\mathbb{M} : (\nabla\mathbf{v}) = \text{trace}(\mathbb{M} * \nabla\mathbf{v}^T)$, $*$ *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 (B.12) by any test function $\mathbf{v}$ (associated to velocity) and multiplying (B.13) by any test function q (associated to pressure), the variational formulation of (B.12) reads, thanks to (B.15),

$$\int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v} + p\nabla \cdot \mathbf{v} + \int_{\partial\Omega} ((-\nabla\mathbf{u} + p\mathbb{I})\mathbf{n}) \cdot \mathbf{v} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \tag{B.16}$$

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 $\partial\Omega$. (B.16) reads now

$$\int_\Omega \nabla\mathbf{u} \colon \nabla\mathbf{v} + p\nabla\cdot\mathbf{v} + \int_{\partial\Omega}((-\nabla\mathbf{u}+p\mathbb{I})\mathbf{n})\cdot\mathbf{v} + \frac{\gamma}{\mathbf{h}}\mathbf{u}\cdot\mathbf{v} = \int_\Omega \mathbf{f}\cdot\mathbf{v} + \int_{\partial\mathbf{\Omega}}\frac{\gamma}{\mathbf{h}}\mathbf{g}\cdot\mathbf{v} \tag{B.17}$$

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

$$\int_\Omega \nabla\mathbf{u} \colon \nabla\mathbf{v} + p\nabla\cdot\mathbf{v} + \int_{\partial\Omega}((-\nabla\mathbf{u}+p\mathbb{I})\mathbf{n})\cdot\mathbf{v} + ((-\nabla\mathbf{v}+q\mathbb{I})\mathbf{n})\cdot\mathbf{u} + \frac{\gamma}{h}\mathbf{u}\cdot\mathbf{v} =$$
$$\int_\Omega \mathbf{f}\cdot\mathbf{v} + \int_{\partial\Omega}((-\nabla\mathbf{v}+q\mathbb{I})\mathbf{n})\cdot\mathbf{g} + \frac{\gamma}{h}\mathbf{g}\cdot\mathbf{v} \tag{B.18}$$

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() );
```

B.4   Stabilisation techniques

B.4.1   *Convection dominated flows*

Consider this type of problem

$$-\epsilon\Delta u + \mathbf{c}\cdot\nabla u + \gamma u = f, \quad \nabla\cdot\mathbf{c} = 0 \tag{B.19}$$

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)

B.4.2   *The CIP methods*

Add the term

$$\sum_{F\in\Gamma_{\mathrm{int}}}\int_F \gamma\, h_F^2\, |\mathbf{c}\cdot\mathbf{n}|\, [\nabla u][\nabla v] \tag{B.20}$$

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

$$[\nabla u] = \nabla u \cdot \mathbf{n}|_1 + \nabla u \cdot \mathbf{n}|_2 \tag{B.21}$$

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

$$[u] = u\mathbf{n}|_1 + u\mathbf{n}|_2 \tag{B.22}$$

Remark 15 (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 , (γ_β 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))));
```

B.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 C

GNU Free Documentation License

GNU Free Documentation License Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

Bibliography

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

[2] Vincent Chabannes, Gonçalo Pena, and Christophe Prud'Homme. High order fluid structure interaction in 2D and 3D. Application to blood flow in arteries. Submitted to Elsevier Journal of Computational and Applied Mathematics (JCAM), February 2012.

[3] Vincent Doyeux, Yann Guyot, Vincent Chabannes, Christophe Prud'Homme, and Mourad Ismail. Simulation of two-fluid flows using a finite element/level set method. Application to bubbles and vesicle dynamics. Submitted to Elsevier Journal of Computational and Applied Mathematics (JCAM), February 2012.

[4] 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.

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

[6] Gonçalo Pena. *Spectral element approximation of the incompressible Navier-Stokes equations in a moving domain and applications*. PhD thesis, Lausanne, 2009.

[7] Gonçalo Pena and Christophe Prud'homme. Construction of a high order fluid-structure interaction solver. *Journal of Computational and Applied Mathematics*, In Press, Accepted Manuscript, 2009.

[8] 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.

[9] 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.

[10] Christophe Prud'Homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samake, and Gonçalo Pena. Feel++: A Computational Framework for Galerkin Methods and Advanced Numerical Methods. Submitted to ESAIM Proc., January 2012.

[11] Benjamin Stamm. *Stabilization strategies for discontinuous Galerkin methods*. PhD thesis, Lausanne, 2008.

[12] Christoph Winkelmann. *Interior penalty finite element approximation of Navier-Stokes equations and application to free surface flows*. PhD thesis, Lausanne, 2007.