Computer Practices: The Finite Element Method with *FreeFem++*

Juan Vicente Gutiérrez Santacreu† and J. Rafael Rodríguez Galván‡

†Departamento de Matemática Aplicada I. Universidad de Sevilla.
juanvi@us.es
‡Departamento de Matemáticas. Universidad de Cádiz.
rafael.rodriguez@uca.es

Doc-Course: "Partial Differential Equations: Analysis, Numerics and Control". Research Unit 3: Numerical Methods for PDEs

April, 2018

Contents

1 Introduction

1.1 The *FreeFem++* environment

FreeFem++[1] is a package for *numerical approximation of the solution of PDE* (partial differential equations), both 2D and 3D, by means of the *Finite Element Method* (FEM). *FreeFem++* is composed of:

- An interpreted **programming Language**:
  - Oriented to (1) fast specification of (linear and steady) PDE problems and (2) resolution of those problems, using the FEM.

[1] `http://www.freefem.org/ff++`

  - Allows easy implementation of complex problems (nonlinear, transient,...)

- An **interpreter** for that language.
  - Programs in FreeFem++ are interpreted (not compiled) at runtime. In this sense, FreeFem++ is a *scripting* language (like Python, Matlab/Octave, Perl, and others).
  - *FreeFem++* is **open source/free software** (GNU GPL license).
  - There are different versions: `FreFem++`, `FreeFem++-nw`, `FreeFem++-mpi`,...

FreeFem++ is mainly developed by **F. Hetch**.

1.2 What can you do with *FreeFem++*?

Let us present two examples, both of them related to the 2D steady Stokes equations (which model viscous incompressible homogeneous fluxes):

$$\begin{cases} -\nu\Delta\mathbf{u} + \nabla p = f \\ \nabla\cdot\mathbf{u} = 0 \\ \quad + \text{boundary conditions}, \end{cases}$$

where the unknowns are $\mathbf{u} = (u, v) : \Omega \to \mathbb{R}^2$ (velocity field of fluid) and $p : \Omega \to \mathbb{R}$ pressure in each point of the domain.

- Video: Stokes/Navier-Stokes equations in Mediterranean sea.
- Why I like numerical simulation (as mathematician): it helps you to understand underlying maths (Video: instability of a numerical scheme).

1.3 Characteristics of the *FreeFem++* Language

- Inspired by **C/C++**.
  - *Similarities*: Syntax, strong typing...

  - *Does not include*: Pointers, object orienting, ...

- **Oriented** to numerical simulation using the finite element method. **Capabilities**:

  - Definition of the **geometry** of a problem and 2D/3D meshing. Although *FreeFem++* is not a CAD/CAE environment and then, for complex geometries, it is necessary to use external tools.
  - Variety of available **finite elements**: P_k–Lagrange, P_1–bubble, P_1 discontinuous, Raviart-Thomas...
  - Flexibility for definition of problems which can be formulated in terms of PDE (and expressed by a **variational formulation**)
  - Automation of the task of **assembling FEM matrices** (involved in underlying FEM linear systems) so that this task is *transparent to the user.*
  - Several algorithms for **resolution of those linear systems**: LU, Cholesky, Crout, CG, GMRES, UMFPACK...
  - Facilities for **post-processing** and **2D/3D visualization**. Although *FreeFem++* no is not specialized in scientific visualization, it can be complemented with external tools for high-quality graphics.
  - Other issues:
    * Excellent documentation, including a **manual** (with a plenty of examples and tutorials): `http://www.freefem.org/ff++/ftp/freefem++doc.pdf`.
    * (Matlab/Octave/Python/Fortran)–like matrix manipulation.
    * Automatic interpolation between meshes, adaptive refinement,...
    * Parallel (with MPI) version available (`FreeFem++-mpi`).

2 Installation and first steps

FreeFem++ is available for download from its web page or from the software center of some operative systems (e.g. Debian or Ubuntu GNU/Linux). Once installed, the *FreeFem++* standard distribution includes an interpreter for execution of code but no editor or integrated environment (with editor, error feedback, syntax highlighting, etc.). User is allowed to choose his/her preferred editor between different possibilities, for instance (see *FreeFem++* manual for details):

- *Crimson Editor* or *Notepad++* on Windows,
- *Fraise editor* on MacOS.
- `Emacs` on GNU/Linux, MacOS or Windows. Note that Emacs is a free/libre general-purpose (not only specialized in *FreeFem++*) and powerful text editor.
- ***`FreeFem-cs`*** on GNU/Linux, MacOS or Windows. Note that *FreeFem-cs* is integrated environment specialized in editing and post-processing *FreeFem++* scripts.

Here we review the configuration of the last two ones.

2.1 Using *FreeFem++* under Emacs

Emacs can be installed form its web page of from the software center of your operative system. A *FreeFem++* mode is available for Emacs, providing:

- Color-coded editor with automatic highlighting of *FreeFem++*.
- Integrated interface for running *FreeFem++* scripts.
- Tracking of compilation errors, linked back to the EDP source code.

For a **easy configuration** from scratch, it suffices to download and unzip *in your home user directory* the file `freefem-emacs-basic-config.zip` which can be found in `https://github.com/rrgalvan/freefem-mode/releases`.

For advanced configuration of *FreeFem++* mode on Emacs, follow the instructions from `https://github.com/rrgalvan/freefem-mode`.

2.2 Using *FreeFem++* under the integrated environment *FreeFem-cs*

FreeFem-cs (*CS* $\leftarrow$ Client/Server) is a multiple-platform package which contains both *FreeFem++* and an integrated environment for *FreeFem++* providing an intuitive interface. It adds to the goodies described for Emacs the following characteristics:

- Integrated graphics area for 2d and 3d.
- Online help including documentation in HTML.

2.2.1 Installation

For installation, you can get your preferred version from the "*Download*" link (`http://www.ann.jussieu.fr/~lehyaric/ffcs/install.php`) and follow the specific instructions for each platform (which consist in only a few steps). For instance:

- *GNU/Linux* and *MacOS*: Decompress the `.tgz` or the `.zip` file in your preferred location (for instance, in the desktop). Run the program *FreeFem++-cs*.
- *Windows*: Execute the installation program and follow usual steps. Once installed, click on the *FreeFem++-cs* icon and start using the application.

2.3 First steps with *FreeFem++*

In the following exercise, we write a very simple FreeFem++ script which plots a simple mesh in the unit square $[0,1] \times [0,1]$. This mesh is defined by the subdivision of $[0,1]$ into to sub-intervals (both in the x axis in the y axis).

Exercise 1.

Write the following code and run it. Test that a graphic is opened and a message is written in the bottom panel.

```
mesh Th = square(2,2); // Declare a mesh object and build it
plot(Th);
cout << "This first mesh was properly drawn" << endl;
```

Former code may result quite familiar to C++ programmers. In particular, notice that commentaries can be introduced by the `//` operator and console output is produced by the familar C++ keywords `cout`, `endl` and the C++ streaming operator `<<`. On the other hand, `mesh` definition resembles C++ object constructors.

When running this example, a window will be opened showing the mesh[2]. Although this window contains no buttons or menus, it can be controlled by the keyboard and by variables passed when plotting. Press '`?`' key for help and for details see the *FreeFem++* manual and 'Plot' page in the FreeFem++ Wiki Manual.

You can also notice that *FreeFem++* produces a verbose output[3], including the whole code that was run and also some extra information and the text which is streamed to the standard output using the keyword '`cout`'.

2.4 A First Realistic Example

In this section we are going to solve a simple elliptic PDE problem by means of the FEM method. More complex problems are left for further sections. Specifically, here we are going to solve the following example (Poisson equation with homogeneous Dirichlet boundary conditions) in a domain $\Omega \subset \mathbb{R}^2$ defined as the unit circle:

$$\begin{cases} \text{Find } u : \overline{\Omega} \to \mathbb{R} \text{ such that} \\ -\Delta u = f \quad \text{in } \Omega, \\ \qquad u = 0 \quad \text{on } \partial\Omega. \end{cases}$$

For that purpose, we proceed as follows:

Step 1. Express the problem in (discrete) variational formulation:

$$\begin{cases} \text{Find } u : \overline{\Omega} \to \mathbb{R} \text{ such that} \\ -\Delta u = f \quad \text{in } \Omega, \\ \qquad u = 0 \quad \text{on } \partial\Omega. \end{cases} \quad \rightsquigarrow \quad \begin{cases} \text{Find } u_h \in V_h \text{ such that} \\ \displaystyle\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f \cdot v_h, \quad \forall v_h \in V_h. \end{cases}$$

Step 2. Translate the variational formulation into *FreeFem++* language. Supposing that the domain, Ω, is given by the unit circle, we can write the following script :

[2] Although in *FreeFem-cs*, windows are opened in the right top panel

[3] Which can be controlled by setting the variable `verb=n`, where `n` indicates the verbosity level. So that `verb=0` sets the lowest verbosity level.

Listing 1: First example: Poisson problem homogeneous Dirichlet conditions

```
1  border gamma(t=0, 2*pi) { x=cos(t); y=sin(t); } // Define boundary
2  mesh Th = buildmesh(gamma(20)); // Build mesh with 20 intervals on
      boundary
3  fespace Vh(Th,P1); // Define P1 finite element space
4  Vh u,v; // Build two functions in this space (unknown and test function)
5  func f = exp(x)*sin(y);  // Right hand side
6  solve Dirichlet(u,v) =   // Define variational formulation
7    int2d(Th) ( dx(u)*dx(v) + dy(u)*dy(v) )
8    - int2d(Th) ( f*v )
9    + on( gamma, u=0 );
10 plot(u);  // Show results
```

This piece of code contains the fundamentals of FEM with *FreeFem++*.

1. In lines 1 and 2 we define the circular domain. The technique consists of the parametrization of the boundary. Any domain with parametrizable boundary can be easily introduced in *FreeFem++*. For other domains, one has to use a specific tool for mesh construction.

2. In line 3 we define the FE (finite element) space, $\mathbb{P}_1$–Lagrange in this case, and in line 4 we declare two variables in this space. We intend to use the first one, `u`, as the FE unknown (the `trial` function), and the second one, `v` as the `test` function.

3. In line 5 we define a function. Note that standard variables `x` and `y` are predefined and must not be declared.

4. In lines 6–9 we solve the variational problem. Note that those lines constitute a quasi-literal transcription of the variational problem formulated in `Step 1`. Some comments:

   (a) By default, PDE operators like gradient (∇) are not predefined (although they can be defined using macros, as we see in a further section). So one must use the operators `dx` (i.e. $\frac{\partial}{\partial x}$) and `dy` (i.e. $\frac{\partial}{\partial y}$). For 3D programs, also `dz` can be employed.

   (b) Dirichlet conditions are imposed as the "artificial" sum of a term to the bilinear form.

5. Finally, in line 10 we plot the obtained solution. Scalar data (as is, in this case, u) is plotted by contour plots, while vector data is plotted as arrow field (for instance, the velocity unknown in the context of Stokes equations).

Exercise 2.

Modify the previous example for solving Poisson equation in the unit square with $f = 0$ and the following boundary conditions: $u = 1$ on the top edge of the square, $u = 0$ on bottom, left and right edges of the square.

Hint: use "mesh Th=square(n,n)" to define a mesh with n subintervals in horizontal and vertical edges. Then boundary edges are labeled as 0 (bottom), 1 (right), 2 (top) and 3 (left).

2.5 Saving to VTK for high-quality graphics

Despite *FreeFem++* includes powerful graphics, it also integrates nicely with data analysis and visualization software through the VTK file format.

VTK consists of an open source C++ library for visualization of different types of data (scalar, vector, tensor, etc.). Last versions of FreeFem++ include a module (called `iovtk`) which can be loaded for use VTK. This way users can save any FE function to a `.vtk` file and then employ any of the available advanced applications for manipulation and visualization of the data contained in that file. In section C we delve into one of those applications, called Paraview[4].

The following code can be appended to the script above for saving the solution, u, into a VTK file.

```
load "iovtk";
savevtk("/tmp/output.vtk", Th, u, dataname="Temperature");
```

The module `iovtk` provides the function `savevtk`. Their compulsory parameters are: (1) name of the output VTK file, (2) name of the mesh, (3) FE function to be saved. More than one function can be saved in the same file, as we will see below. The last parameter is optional (but recommended) and provides a name for each saved function. In this case, assuming that the solution represents the equilibrium state of a heating experiment, the only data set is called "Temperature". We can use this name to access the data in the future (for instance using Paraview).

3 Other Boundary Conditions

In this section we go beyond the Poisson problem and generalize it for different boundary conditions:

1. Neumann boundary conditions.
2. Robin boundary conditions.

3.1 Poisson Problem With Mixed Neumann/Dirichlet Boundary Conditions

Let us consider the following problem: given

- $\Omega \subset \mathbb{R}^2$, with smooth piecewise boundary, divided into two non-empty subsets: $\partial\Omega = \Gamma_D \cup \Gamma_N$,
- $\nu > 0$, $f : \Omega \to \mathbb{R}$, $g_D : \Gamma_D \to \mathbb{R}$ and $g_N : \Gamma_N \to \mathbb{R}$,

[4] http://www.paraview.org/

find $u : \overline{\Omega} \to \mathbb{R}$ such that:

$$\begin{cases} -\nu \Delta u = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma_D, \\ \dfrac{\partial u}{\partial n} = g_N & \text{on } \Gamma_N. \end{cases} \tag{1}$$

Notice that we consider a Dirichlet b.c. in Γ_D and a Neumann b.c on Γ_N. Remember that last condition means, means $\nabla u \cdot n = g_N$, where n is the exterior normal vector.

- The theory for **non-homogeneous Dirichlet** conditions, $u|_{\Gamma_D} = g_D$, is based on writing the solution as

  $$u = u_0 + u_D,$$

  where u_0 is a solution of the homogeneous problem ($u|_{\Gamma_D} = 0$) and u_D verifies $u|_{\Gamma_D} = g_D$. Algorithms for implementing generic Dirichlet conditions requires some kind of computing technique[5] which is automatized by *FreeFem++* (just the operator '`on( )`' is used as in previous examples) and then we are not going deeper.

- Moreover **Neumann boundary conditions** appear in a natural way in the variational formulation. Specifically, when the Green formula (integration by parts) is applied, one gets the following problem:

  $$\begin{cases} \text{Find } u_h \in V_h \text{ such that} \\ \nu \displaystyle\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f\, v_h + \nu \int_{\Gamma_N} g_N\, v_h \quad \text{for all } v_h \in V_h, \end{cases}$$

  where for a fixed polynomial degree $k \in \mathbb{N}$,

  $$V_h = \big\{ v_h \in C^0(\Omega) \text{ such that } v_h|_T \in \mathbb{P}_k[x]\ \forall T \in \mathcal{T}_h \text{ and } v_h|_{\Gamma_D} = 0 \big\}.$$

Here we show a FreeFem++ program for the Poisson problem presented above. Here we use the keyword `problem` for defining the variational problem, which is solved later (instead of `solve`, which was used in Example 1 for defining and solving the problem).

Listing 2: Poisson problem with mixed Dirichlet/Neumann boundary conditions

```
// 1. Pre-processing

// 1.1. Mesh
border gamma0(t=2*pi, 0) { x=1.5*cos(t); y=sin(t); }
border gamma1(t=0, 2*pi) { x=4*cos(t); y=4*sin(t); }
mesh Th = buildmesh(gamma1(40)+gamma0(30));
```

[5]In practice, for Dirichlet conditions one proceed as follows:

1. Build the FE linear system $Ax = b$, where A comes from a bilinear form, $a(\cdot, \cdot)$ and b comes from the linear form, $L(\cdot)$. Both of A and b are, typically constructed by quadrature formulae in triangles.
2. Select the rows of A and b which correspond to equations for degrees of freedom on Γ_D. Then modify them, imposing explicitly the value of u on those degrees of freedom. Typically, just the diagonal element and the RHS (containing the boundary value) are multiplied by a Huge number.

```
plot(Th, wait=1);

// 1.2. FE space and functions
fespace Vh(Th,P1);
Vh u,v;

// 1.3. Definition of data
real nu=0.3;
func f=0;
func g0=4;
func g1=1;

// 2.- Defining the problem and solving it

problem PoissonDirNeu(u,v) =
  // Bilinear form:
  int2d(Th)( nu*( dx(u)*dx(v) + dy(u)*dy(v) ))
  // Linear form:
  - int2d(Th)( f*v )
  - int1d(Th, gamma1)( nu*g1*v )
  // Dirichlet boundary condition
  + on(gamma0, u=g0);

PoissonDirNeu; // Mount linear system and solve the problem

// 3. Post-processing
plot(u, value=1, fill=1, wait=1);
```

4 Delving into the FreeFem++ language

4.1 Data Types, Arrays and Matrices

Fundamental data types are similar to C++. But some fundamental types of C++ are not present in FreeFem++ (and vice versa) for instance:

Listing 3: FreeFem++ fundamental data types and operations

```
// 1) Integers

int i;
i = 10;
int j =-10;
// unsigned k=20; // Error the identifier usigned does not exist
cout << i << ", " << j << endl;

// Some arithmetics
cout << " i+j:" << i+j << endl;
cout << "max(i,j): " << max(i,j) << endl;
cout << "square(i): " << i^2 << " or " << square(i) << endl;
cout << "sqrt(abs(j)): " << sqrt(abs(j)) << endl;

// 2) Floatting point numbers
```

```

real x, y; // Double precisson numbers (termed double in C++)
x = 0.001;
y = pi; // Pi is a pre-defined keyword

// 3) Also complex
complex c = 1-2i;
cout << "c=" << c << "... " << 2*c-2 << endl;

// 4) Characters and strings
// char s = 'a' // // Error the identifier char does not exist
string s = "a";
string t = "Esto es una cadena de caracteres";
cout << s << ", " << t << endl;
cout << "Uni\'on " << ", " << s+t << endl;
cout << "Subconjunto: " << t(0:3) << endl;

// 5) Arrays
cout << endl << endl;

real [int] v(10);  // array of 10 real
v = 1.03; // set all the array to 1.03
v[1]=2.15;
cout << v[0] << " " << v[9] << " size of v = "
  << v.n << " min: " << v.min << " max:" << v.max
  << " sum : " << v.sum << endl;
// change the size of array
v.resize(12);
v(10:11)=3.14;
v(5:9)=sqrt(2);
cout << " resized v: " << v << endl;
real[int] w1(12), w2(12);
w1 = 2*v;
w2 = w1+v;
cout << "w2:" << w1 << endl;
cout << "min: " << v.min << ", sum: " << v.sum << endl;

// Arrays with two indexes

int N=3;
real[int,int] A(N,N); // Squared NxN matrix
real[int] b1(N), b2(N);
b1=[4,5,6];
b2=[1,2,3];
A=1; // Fill A with ones
A(:,1)=2; // Fill first column
cout << A << endl;
cout << b1'*A << endl; // b^T times A
cout << b1'*b2 << endl; // Scalar product

// Sparse matrices

matrix M = A; // Now M is a sparse matrix
cout << "Storage: for each nozero value, row column value(row, column)" <<
```

```
    endl;
cout << M << endl;
```

4.2 Linear System Associated to FEM

The keyword **varf** can be used in *FreeFem++* to store the matrix and vector related to a variational formulation. Then the operator `^-1*` can be used to solve the associated linear system. The advantage of using the **varf** approach is that, if used properly, it is faster that `solve` or `problem` (about 4 times faster, according to *FreeFem++* documentation).

The following script uses **varf** for solving Example 2.

Listing 4: Linear System Associated to Variational Formulation

```
// 1. Pre-proceso

// 1.1. Mesh
mesh Th = square(4, 4);
plot(Th, wait=1);
int[int] dirichletBoundary = [1,2,3,4];

// 1.2. FE space and functions
fespace Vh(Th,P1);
Vh u,v;

// 1.3. Definition of data
real nu=1;
func f=4;
func g=0;

// 2.- Defining the problem

varf PoissonDirichletMatrix(u,v) =
  // Bilinear form:
  int2d(Th)( nu*( dx(u)*dx(v) + dy(u)*dy(v) ))
  // Dirichlet boundary condition
  + on(dirichletBoundary, u=g);

varf PoissonDirichletRHS(u,v) =
  // Linear form:
  int2d(Th)( f*v )
  // Dirichlet boundary condition
  + on(dirichletBoundary, u=g);

matrix A = PoissonDirichletMatrix (Vh, Vh); // Mount sparse matrix
real[int] b = PoissonDirichletRHS(0, Vh); // Mount RHS

u[] = A^-1 * b; // Solve the linear system and store vector into u

// 3. Post-processing
plot(u, value=1, fill=1, wait=1);
```

Exercise 3.

Compare computing time of `problem` and `varf` approaches for solving the Poisson problem with Dirichlet conditions. Search FreeFem++ documentation for find a function which can be utilized for calculation of the computing time. *Hint: use the function* `clock( )` *for getting current time, then compute time difference.*

5 Solving Evolution Equations

5.1 The Implicit Euler Method for the Heat Equation

Former examples were steady, namely time independent. In this section we are going to solve the following transient problem (heat equation with mixed Dirichlet+Neumann boundary conditions). Of course, although the solution is going to be called "temperatura", it is applied to any parabolic diffusive law (e.g. in biological models). The problem reads: find $u = u(x,t)$ such that

$$\begin{cases} \dfrac{\partial u}{\partial t} - \nu \Delta u = f \quad \text{in } \Omega, \\ \qquad u(0) = u_0 \text{ in } \Omega, \\ \qquad\quad u = g_D \quad \text{on } \Gamma_D, \\ \qquad \dfrac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N. \end{cases} \tag{2}$$

Here:

- $x \in \Omega \subset \mathbb{R}^2$, with smooth piecewise boundary, $\partial\Omega = \Gamma_D \cup \Gamma_N$
- $t \in [0,T]$, where $T > 0$ is the final time
- u_0: $\Omega \to \mathbb{R}$: temperature at initial time.
- $\nu > 0$, $f : \Omega \times (0,T) \to \mathbb{R}$ (heat source in the domain), $g_D : \Gamma_D \times (0,T) \to \mathbb{R}$ (heat source on Dirichlet boundary Γ_D), $g_N : \Gamma_N \times (0,T) \to \mathbb{R}$ (heat source on Neumann boundary Γ_N).

For time discretization, we fix $n \in \mathbb{N}$ and consider the following $n+1$ time instants in $[0,T]$:

$$t_m = \Delta t \cdot k, \ m = 0, ..., N$$

where $\Delta t = T/N$ is the time step.

The *Implicit Euler* method for (2) can be written as follows: Given $u_h^0 = u_0$, for each $m = 0, ..., N-1$, find $u_h^{m+1} \in U_h$ (space defined in section 3.1) such that:

$$\begin{cases} \dfrac{u_h^{m+1} - u_h^m}{\Delta t} - \nu \Delta u_h^{m+1} = f_h^{m+1} \quad \text{in } \Omega, \\ \qquad\qquad u_h^{m+1} = g_D^{m+1} \quad \text{on } \Gamma_D, \\ \qquad\qquad \dfrac{\partial u_h^{m+1}}{\partial n} = g_N^{m+1} \quad \text{on } \Gamma_N. \end{cases} \tag{3}$$

Former problem can be written in variational formulation as follows:

$$a(u_h^{m+1}, v_h) = b(v_h) \quad \forall v_h \in U_h,$$

where

$$a(u,v) = \frac{1}{\Delta t}\int_\Omega u^{m+1} v + \nu \int_\Omega \nabla u^{m+1} \cdot \nabla v,$$

$$b(v) = \int_\Omega f \cdot v + \frac{1}{\Delta t}\int_\Omega u^m v + \int \Gamma_N \nu \int_\Omega g1 v.$$

5.1.1 *FreeFem++* program

For instance, the following script shows the resolution of heat equation by implicit Euler time scheme (3), for the following data:

- Domain Ω: unit circle. $\partial\Omega$ is split into two boundaries: Γ_D is defined by angle in $(0, \pi/4)$ while Γ_D is defined by angle in $(\pi/4, 2\pi)$. Space discretization is defined by 100 sub-intervals approximating $\partial\Omega$.
- Time interval: $[0, T] = [0, 1]$, with time step $\delta_T = 1/100$.
- Initial boundary condition: $u(x, 0) = 0$ in Ω. Dirichlet boundary condition: $g_D(x,t) = 100 - 100/(t+1)$ on $\Gamma_D \times (0, T)$. Neumann boundary condition: $g_N(x,t) = 0$ on $\Gamma_N \times (0, T)$.
- $f = 0$, viscosity parameter $\nu = 1$.

Listing 5: Heat Equation

```
load "iovtk"; // We will output vtk

// 1. Pre-processing

// 1.1. Mesh
real R=1;
border gammaD(t=0, pi/4)    { x=R*cos(t); y=R*sin(t); }
border gammaN(t=pi/4, 2*pi) { x=R*cos(t); y=R*sin(t); }

int n=10;
mesh Th = buildmesh(gammaD(n)+gammaN(9*n));
plot(Th, wait=1);

// 1.2. FE space and functions
fespace Vh(Th,P1);
Vh u, v;
Vh uold;

macro gradient(u) [dx(u), dy(u)] // End Of Macro

// 1.3. Data definition
real nu=1;
real t=0, T=1;  // Time interval [0,T]
int  N=100; // Number of time iterations
```

```
real dt=T/N;  // Time step

func f=0;
func real gD(real x, real y, real t) {
  return 100*(1-1./(t+1));
}
func real gN(real x, real y, real t) {
  return 0;
}

func u0=0; // Init

uold = u0;

// 2. Processing

// Declare (but not solve) the heat equation variational problem
problem heatEquation(u,v)=
  // Bilineal form:
  int2d(Th)(
    u*v/dt +
    nu*gradient(u)'*gradient(v) // ' means transpose
  )
  // Linear form
  - int2d(Th)( uold*v/dt + f*v )
  - int1d(Th, gammaN) ( gN(x,y,t)*v ) // Neumann boundary condtion

  // Dirichlet boundary condtion
  + on(gammaD, u=gD(x,y,t));

// Time iteration loop
for (int k=0; k<N; ++k ) {

  t = t + dt;     // Increase current time
  heatEquation;  // Solve the PDE variational problem
  uold = u;      // Save solution for next time step

  plot(Th, u, fill=true, value=true);

  // 3. Post-processing (save to VTK for further displaying with Paraview)
  string filename="/tmp/heat_equation-" + k + ".vtk";
  savevtk(filename, Th, u, dataname="Temperature");
}
```

A Poisson Problem With Robin Boundary Conditions

Let us consider the following PDE problem, which generalizes (1):

$$\begin{cases} -\nu\Delta u = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma_D, \\ au + b\dfrac{\partial u}{\partial n} = g_N & \text{on } \Gamma_N, \end{cases} \tag{4}$$

where $a, b \in \mathbb{R}$. For $a, b \neq 0$, the third equation in (4) is termed a *Robin boundary condition.*

Integration by parts one can obtain the following variational formulation:

$$\begin{cases} \text{Find } u_h \in U_h \text{ such that} \\ \nu \displaystyle\int_\Omega \nabla u_h \cdot \nabla v_h + \nu\frac{a}{b}\int_{\Gamma_N} uv = \int_\Omega f\, v_h + \nu\frac{1}{b}\int_{\Gamma_N} g_N\, v_h \quad \text{for each } v_h \in U_h, \end{cases}$$

being U_h defined as above.

Exercise 4.

Develop a FreeFem++ script for the finite element approximation of the solution of the problem presented above. For instance, use $a = 1, b = 1$ and the same domain and data as in Listing 2.

B The Stokes equations

The Stokes equations can be considered as the linear steady version of Navier-Stokes equations (which describe the behaviour of a newtoninan fluid as atmosphere, ocean, flux around vehicles, etc.

$$\begin{cases} -\nu\Delta \mathbf{u} + \nabla p = f \\ \nabla \cdot \mathbf{u} = 0 \\ \qquad + \text{boundary conditions}, \end{cases}$$

where the unknowns are: $\mathbf{u} = (u, v) : \Omega \to \mathbb{R}$ (velocity field of fluid) and $p : \Omega \to \mathbb{R}$ pressure in each point of the domain. Thus the first equation must be understood in vectorial way, specifically, in the 2D case:

$$\Delta u + \partial_x p = f_1,$$
$$\Delta v + \partial_y p = f_2,$$

where $f = (f_1, f_2)$.

In this section we show a usual test for the Stokes 2D simultion, which is know as **cavity test**. This test is usually run in a rectangular domain but, in this case, with the purpose of illustrate the construction in *FreeFem++* of complex parametric geometries, we have introduced some holes in the rectangular domain. They are defined by parametric figures which are known as conchoids[6].

[6] http://en.wikipedia.org/wiki/Conchoid_%28mathematics%29

Homogeneous Dirichlet b.c., $(u, v) = (0, 0)$, are imposed for $\mathbf{u}$ on the whole boundary excepting the top line, where we fix $(u, v) = (1, 0)$ (positive horizontal velocity). We use the stable FE combination $\mathbb{P}_2/\mathbb{P}_1$ (polynomials with degree 2 for velocity and degree 1 for pressure).

B.1 *FreeFem++* Programming of 2D Stokes System

Listing 6: Steady Stokes System

```
// 2D Stokes equations
// Cavity test in a domain with some parametric holes

//,-----------------------------------------------
//| STEP 1. Defining the domain and meshing it
//`-----------------------------------------------

// Macro for the 2D boundary defining a hole. They are parametric
// curves called "conchoids". In the macro:
// n = number of 'petals', P = center of the hole
int NMAX=20;
macro conchoid(name, n, P, thelabel)
 name(i=0,NMAX) {
    real a=1.0, b=2.0;
    real theta = i*2*pi/NMAX;
    real rho = a * cos(n*theta)+b;
    x = P[0] + rho*cos(theta);
    y = P[1] + rho*sin(theta);
    label = thelabel;
} // EOM

// Definition of some conchoids
border conchoid(c2,2,[0,0]   ,0);
border conchoid(c3,3,[-10,0] ,0);
border conchoid(c4,4,[0,0]   ,0);
border conchoid(c5,5,[10,0]  ,0);
border conchoid(c6,6,[0,0]   ,0);
border conchoid(c7,7,[0,0]   ,0);

// External rectangle
real xcoor = 15, ycoor = 5;
border lx1(k=-xcoor,xcoor) { x=k; y=-ycoor; label=1; }
border lx2(k=-xcoor,xcoor) { x=k; y=+ycoor; label=3; }
border ly1(k=-ycoor,ycoor) { x=-xcoor; y=k; label=2; }
border ly2(k=-ycoor,ycoor) { x=+xcoor; y=k; label=2; }

int nx=40, ny=20, nc=50;
mesh Th = buildmesh( ly1(-ny)+lx1(nx)+ly2(ny)+lx2(-nx)
    + c3(-nc) + c4(-nc) + c5(-nc) );

//,-----------------------------------------------------
//| STEP 2. Resolution of Stokes problem in previous domain
//`-----------------------------------------------------

fespace Uh(Th,P2); Uh u,v,uu,vv;  // Velocity functions
```

```
fespace Ph(Th,P1); Ph p,pp;        // Pressure functions

real upperVelocity=1;

macro grad(u) [dx(u), dy(u)] // end of macro

// Definition of Stokes problem

problem stokes2d( [u,v,p], [uu,vv,pp], solver=LU) =
    int2d(Th)(
        grad(u)'*grad(uu) + grad(v)'*grad(vv)
        + grad(p)'*[uu,vv] + pp*(dx(u)+dy(v)) //'
        - 1e-10*p*pp )
  + on(0,1,2,u=0,v=0) + on(3,u=upperVelocity,v=0);

stokes2d;  // Resolution of Stokes problem

// Save to VTK (for high quality plotting)
load "iovtk";
savevtk("/tmp/stokes.vtk", Th, [u,v,0], p);
```

C Paraview

ParaView is an open source multiple-platform application for interactive, scientific visualization. It was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.

For visualization of data, that lives in a mesh where the simulation was performed, there are basically three steps:

1. *Reading* data into Paraview (from a VTK file)
2. *Filtering*, that is applying one or more filters in order to generate, extract or derive features from data.
3. *Rendering* an image from the data and adjusting the viewing parameters for improve the final visualization.

This tree steps are controlled through a panel in the right, called `Pipeline browser`. The pipeline concept consists on a chain of modules, starting from the data stored in a file. Each of them takes in some data, operates on it and presents the result in a dataset. From the Paraview users guide:

> "Reading data into ParaView is often as simple as selecting `Open` from the *File* menu, and then clicking the glowing *Accept* button on the reader's *Object Inspector* tab. ParaView comes with support for a large number of file formats, and its modular architecture makes it possible to add new file readers. Once a file is read, ParaView automatically renders it in a view. In ParaView, a view is simply a window that shows data. There are different types of views, ranging from qualitative computer graphics rendering of the data to quantitative

spreadsheet presentations of the data values as text. ParaView picks a suitable view type for your data automatically, but you are free to change the view type, modify the rendering parameters of the data in the view, and even create new views simultaneously as you see fit to better understand what you have read in. Additionally, high-level meta information about the data including names, types and ranges of arrays, temporal ranges, memory size and geometric extent can be found in the *Information* tab."

Advanced data processing can be done using the Python Programmable filter with VTK, NumPy, SciPy and other Python modules.

For further details:

1. Video showing how to use FreeFem++ and Paraview for visualization of 2D and 3D cavity tests for the Stokes Equations (partially in spanish). `https://www.youtube.com/watch?v=wChDeo2A03E`

2. Paraview Wikipedia page (in which this appendix is based). `http://en.wikipedia.org/wiki/ParaView`.

3. Resources in the web, for instance `http://vis.lbl.gov/NERSC/Software/paraview/docs/ParaView.pdf`.

4. The paraview users guide (how to unleash the beast!) `http://denali.princeton.edu/Paraview/ParaViewUsersGuide.v3.14.pdf`