

FREEFEM User Manual

A language for the finite element method

Olivier Pironneau¹
Christophe Prud'homme²

Revised: 10/20/01

¹`pironneau@ann.jussieu.fr`; Laboratoire d'analyse numérique; Université
Pierre et Marie Curie; 75005 Paris France

²`prudhomm@mit.edu` Mechanical Engineering Department; Massachussetts Insti-
tute Of Technology; 77, Mass Ave Room 3-264; 02139 Cambridge USA

Copyright (c) 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001 Christophe Prud'homme and Olivier Pironneau.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

1	Introduction	4
1.1	Conventions	4
1.2	Software and Documentation	5
2	Installation	6
2.0.1	Configure script	6
3	The Gory Details	8
3.1	Programs	8
3.2	List of reserved words	9
3.3	Building a mesh	10
3.3.1	Triangulations	10
3.3.2	Border(), buildmesh(), polygon()	10
3.3.3	Geometric variables, inner and region bdy, normal	12
3.3.4	Regions	13
3.4	Functions	13
3.4.1	Functions and scalars	13
3.4.2	Building functions	15
3.4.3	Value of a function at one point	15
3.4.4	Special functions:dx(), dy(), convect()	16
3.5	Global operators	17
3.6	Integrals	17
3.7	Solving an equation	19
3.7.1	Onbdy()	19
3.7.2	Pde()	19
3.7.3	Solve()	20
3.7.4	2-Systems	21
3.7.5	Boundary conditions at corners	22
3.7.6	Weak formulation	22

3.8	Results	24
3.8.1	plot, savemesh, save, load, loadmesh	24
3.8.2	Saveall()	26
3.9	Other features	28
3.9.1	Iter	28
3.9.2	Complex numbers	28
3.9.3	Scal()	29
3.9.4	Wait, nowait, changewait	29
3.9.5	One Dimensional Plots	29
3.9.6	Precise	30
3.9.7	Exec(), user(), how to link an external function to Gfem	30
3.10	Language internals	31
4	Examples	33
4.1	Triangulations examples	33
4.2	Scalar examples	34
4.3	Complex number example	38
4.4	2-system example	38
5	GNU Free Documentation License	40
5.1	Applicability and Definitions	41
5.2	Verbatim Copying	42
5.3	Copying in Quantity	42
5.4	Modifications	43
5.5	Combining Documents	45
5.6	Collections of Documents	46
5.7	Aggregation With Independent Works	46
5.8	Translation	46
5.9	Termination	47
5.10	Future Revisions of This License	47

Chapter 1

Introduction

FREEFEM is an implementation of the **Gfem** language dedicated to the finite element method. It provides you a way to solve Partial Differential Equations (PDE) simply. Although you can solve quite complicated problems can be solved also.

In this manual we are going to describe the FREEFEM package:

- Compilation step
- How to use the language
- A quick presentation of how it is done

1.1 Conventions

- A **triangulation**, T , is a set of triangles covering a domain referred later as Ω
- Each **vertex** has a number **iv**, a region reference number **region**, and a boundary reference number **ib** which is zero for internal points.
- An **array-function** is an array of values on the vertices of T . It represents a piecewise linear continuous (or discontinuous if **precise** is used) function on Ω .
- In an integrated environment **Gfem** has the notion of Project: files shown on the desktop as documents with an icon like a triangulated rectangle (on the Macintosh).
- A **project** contains a program, the last triangulation created and one array function, namely the last function displayed.

1.2 Software and Documentation

FREEFEM is available on Internet. Check the following http site:

kfem.sourceforge.net

The tarball uses the following naming convention: `freefem-x.x.x.tar.gz`

Others documentations are available on the FREEFEM Web site. These might provide further enlightenment on the software.

AUTHORS The authors are, in alphabetical order:

- Dominique Bernardi <??>
- Frederic Hecht <Frederic.Hecht@inria.fr>
- Castro J. Manollo <castro@gamba.cie.uma.es>
- Pascal Parole <pascal@area-mundi.fr>
- Olivier Pironneau <pironneau@ann.jussieu.fr>
- Christophe Prud'homme <prudhomm@users.sourceforge.net>

If you are interested in FREEFEM have a look at the web site kfem.sourceforge.net. You can subscribe to various mailing lists and get the latest stuff from the FREEFEM world.

Chapter 2

Installation

The installation steps differs with the operating system. **FreeFEM** supports two kinds systems :

- Windows/Cygwin¹ environment
- Linux/Unix systems

Both systems follows the POSIX standard. It is strongly recommended to use the `configure` script. It is much easier and should be platform independent, you don't have to know anything about your system. Everything is done for you. However one suggests the use of `make` for the GNU Project: it is the best `make` utility and will work without problems with **freefem**. This utility is available at several ftp sites in France or in the USA.

- prep.mit.ai.edu
- ftp.lip6.fr

2.0.1 Configure script

type:

```
configure
make
make install #if you want to install the software on a site
```

The binary **freefem** will be in the directory `src`.

¹<http://www.cygwin.com>

We use **autoconf** and **automake** in conjunction in order to be as general as possible. These tools are not required to use or compile **freeFEM** but if you want to change the code you will need these.

These tools are available on the **GNU** ftp site: `ftp://prep.ai.mit.edu`. You can get them in France at `ftp://ftp.ibp.fr/pub/gnu`.

Type:

```
configure
```

```
make
```

```
make install #if you want to install the software on a site
```

Debugging By default freefem is compiled with the `-O2` flags for optimization. If you want to debug the library, then USE the following **configure** option `--enable-debug`. That is to say, type

```
configure --enable-debug
```

at the **configure** step of the installation.

Chapter 3

The Gory Details

3.1 Programs

Gfem is a small language which generally follows the syntax of the language Pascal. See the list below for the reserved words of this language.

The reserved word **begin** can be replaced by **{** and **end** by **}**.

C programmers: caution the syntax is "...};" while most C constructs use "...;}"

Example 1: Triangulates a circle and plot $f = x * y$

```
1 border(1,0,6.28,20)
2 begin
3   x:=cos(t);
4   y:=sin(t);
5 end;
6 buildmesh(200);
7 f=x*y; plot(f);
```

Example 2: on the circle solve the Dirichlet problem

$-\Delta(u) = x * y$ with $u = 0$ on $\partial\Omega$

```
1 border(1,0,6.28,20)
2 begin
3   x:=cos(t);
4   y:=sin(t);
5 end;
6 buildmesh(200);
```

```

7 solve(u) begin
8   onbdy(1) u =0;
9   pde(u) -laplace(u) = x*y ;
10 end;
11 plot(u);

```

3.2 List of reserved words

Keywords	Explanations
begin, {	Begin a new block
end, }	End a block
if, then, else, or, and, iter	Conditionnals and Loops
x,y,t,ib,iv,region,nx,ny	Mesh variables
log, exp, sqrt, abs, min, max sin, asin, tan, atan, cos, acos cosh, sinh, tanh	Mathematical Functions
I, Re, Im complex	Complex Numbers Enter Complex Number Mode
buildmesh, savemesh, loadmesh, adaptmesh	Mesh related Functions build, save , load and mesh adaptation
one, scal, dx, dy, dxx, dyy, dxy, dyx, convect	Mathematical operators can be called wherever you want
solve pde, id, dnu laplace, div onbdy plot, plot3d	Enter Solver Mode Solver Functions graphical functions: plot isolines in 2D and Elevated Surface in 3D
save, load, saveall	Saving and Loading Functions
wait, nowait, changewait	Change the Wait State: if wait then the user must click in the window to continue
precise, halt, include, evalfct, exec, user	Miscellaneous Functions

Table 3.1: Gfem Keywords

3.3 Building a mesh

3.3.1 Triangulations

To create a triangulation you must either

- Open a project
- Read an old triangulation stored in text format
- Execute a program which contains the keyword **buildmesh**
- Create one by hand drawing the boundary of Ω and activate the menu **Triangulate** (Macintosh only).

In integrated environments, once created, triangulations can be displayed, stored on disk in **Gfem** format or in text format or even a zoom of its graphic display can be stored in Postscript format (to include it in a TeX file for example).

Gfem stores for each vertex its number and boundary identification number and for each triangle its number and region identification number. Edges number are not stored, but can be recovered by program if needed.

3.3.2 **Border()**, **buildmesh()**, **polygon()**

Use it to triangulate domain defined by its boundary. The syntax is

```
1 border(ib,t_min,t_max,nb_t)
2   begin
3     ...x:=f(t);
4     ...y:=g(t)...
5   end;
6 buildmesh(nb_max);
```

where each line with **border** could be replaced by a line with **polygon**

```
1 polygon(ib,'file_name'[,nb_t]);
```

where **f,g** are generic functions and the [...] denotes an optional addition. The boundary is given in parametric form. The name of the parameter must be **t** and the two coordinates must be **x** and **y**. When the parameter goes from **t_min** to **t_max** the boundary must be scanned so as to have Ω on its left, meaning counter clockwise if it is the outer boundary and

clockwise if it is the boundary of a hole. Boundaries must be closed but they may be entered by parts, each part with one instruction **border** , and have inner dividing curves; **nb_t** points are put on the boundary with values $t = t_{min} + i * (t_{max} - t_{min}) / (nb_t - 1)$ where **i** takes integer values from 0 to **nb_t-1** .

The triangulation is created by a Delaunay-Voronoi method with **nb_max** vertices at most. The size of the triangles is monitored by the size of the nearest boundary edge. Therefore local refinement can be achieved by adding inner artificial boundaries.

Triangulation may have boundaries with several connected components. Each connected component is numbered by the integer **ib** .

Inner boundaries (i.e. boundaries having the domain on both sides) can be useful either to separate regions or to monitor the triangulation by forcing vertices and edges in it. They must be oriented so that they leave Ω on their right if they are closed. If they do not carry any boundary conditions they should be given identification number **ib=0** .

The usual **if... then ... else** statement can be used with the compound statement: **begin...end** . This allows piecewise parametric definitions of complicated or polygonal boundaries.

The boundary identification number **ib** can be overwritten. For example:

```

1 border(2,0,4,41) begin
2   if(t<=1)then { x:=t; y:=0 };
3   if((t>1)and(t<2))then { x:=1; y:=t-1; ib=1 };
4   if((t>=2)and(t<=3))then { x:=3-t; y:=1 };
5   if(t>3)then { x:=0; y:=4-t }
6 end;
7 buildmesh(400);

```

Recall that **begin** and **{** is the same and so is **end** and **}**. Here one side of the unit square has **ib=1**. The 3 other sides have **ib=2**.

The keyword **polygon** causes a sequence of boundary points to be read from the file **file_name** which must be in the same directory as the program. All points must be in sequential order and describing part of the boundary counter clockwise; the last one should be equal to the first one for a closed curve.

The format is

```

1 x[0]    y[0]
2 x[1]    y[1]

```

```
3 | . . . .
```

each being separated by a tab or a carriage return. The last parameter `nb_t` is optional; it means that each segment will be divided into `nb_t+1` equal segments (i.e. `nb_t` points are added on each segments).

For example

```
1 | polygon(1, 'mypoints.dta', 2);
2 | buildmesh(100);
```

with the file `mypoints.dta` containing

```
1 | 0.      0.
2 | 1.      0.
3 | 1.      1.
4 | 0.      1.
5 | 0.      0.
```

triangulates the unit square with 4 points on each side and gives `ib=1` to its boundary. Note that `polygon(1, 'mypoints.dta')` is like `polygon(1, 'mypoints.dta', 0)`.

buildmesh and domain decomposition

There is a problem with `buildmesh` when doing domain decomposition: by default **Gfem** swap the diagonals at the corners of the domain if the triangle has two boundary edges. This will lead to bad domain decomposition at the sub-domain interfaces.

To solve this, there is a new flag for `buildmesh` which is optional:

```
buildmesh(<max_number_of_vertices>, <flag>)
      where <flag> =
```

$$\begin{cases} = 0 & \text{classic way: do diagonal swaping} \\ = 1 & \text{domain decomposition: no diagonal swaping} \end{cases}$$

3.3.3 Geometric variables, inner and region bdy, normal

- `x,y` refers to the coordinates in the domain
- `ib` refers to the boundary identification number; it is zero inside the domain.

- **nx** and **ny** refer to the x-y components of the normal on the boundary vertices; it is zero on all inner vertices.
- **region** refers to the domain identification number which is itself based on an internal number, **ngt**, assigned to each triangle by the triangulation constructor.

Inner boundaries which divide the domain into several simply connected components are useful to define piecewise discontinuous functions such as dielectric constants, Young modulus...

Inner boundaries may meet other boundaries only at their vertices. Such inner boundaries will split the domain in several sub-domains.

3.3.4 Regions

A sub-domain is a piece of the domain which is simply connected and delimited by boundaries.

Each sub-domain has a **region** number assigned to it. This is done by **Gfem**, not by the user. Every time **border** is called, an internal number **ngt** is incremented by 1. Then when the key word **border** is invoked the last edge of this portion of boundary assigns this number to the triangle which lies on its left. Finally all triangles which are in this subdomain are reassigned this number.

At the end of the triangulation process, each triangle has a well defined number **ngt**. The number **region** is a piecewise linear continuous interpolation at the vertices of **ngt**. To be exact, the value of **region** at a vertex (x_0, y_0) is the value of **ngt** at $(x_0, y_0 - 10^{-6})$, except if **precise** is set in which case **region** is equal to **ngt**.

3.4 Functions

3.4.1 Functions and scalars

Functions are either read or created.

- Functions can be read from a file if its values at the vertices of the triangulation are stored in text format. (Open a .dta example with a text editor to see the format).
- Functions can be created by executing a program. An instruction like **f=x*y** really means that $f(x, y) = x * y$

for all x and y . Here x and y refer to the coordinates in the domain represented by the triangulation.

- Functions can be created with other previously defined functions such as in `g=sin(x*y); f=exp(g);` .
- Four other variables can be used besides x , y , iv , t : `nx`, `ny`, `ib`, `region`.

Most usual functions can be used:

```
1 max, min, abs, atan, sqrt,
2 cos, sin, tan, acos, asin, one,
3 cosh, sinh, tanh, log, exp
```

`one(xy|0)+` for instance means 1 if `xy|0+` and 0 otherwise.

Operators:

```
1 and, or, < , <=, < , >=, ==, +, -, *, /, ^
2 x^2 means x*x
```

Functions created by a program are displayed only if the key word `plot()` or `plot3d()` is used (here `plot(f)`).

Derivatives of functions can be created by the keywords `dx()` and `dy()`

Unless `precise` is set, they are interpolated so the results is also continuous piecewise linear (or discontinuous when `precise` is set). Similarly the convection operator `convect(f,u1,u2,dt)` defines a new function which is approximately

$$f(x - u1(x,y)dt, y - u2(x,y)dt)$$

Scalars are also helpful to create functions. Since no data array is attached to a scalar the symbol `:=` is useful to create them, as in

```
1 a:= (1+sqrt(5))/2;
2 f= x*cos(a*pi*y);
```

Here `f` is a function, `a` is a scalar and `pi` is a (predefined) a scalar.

It is possible to evaluate a function at a point as in `a:=f(1,0)` Here the value of `a` will be 1 because `f(1,0)` means `f` at $x=1$ and $y=0$.

3.4.2 Building functions

There are 6 predefined functions: `x,y,ib,region, nx, ny` .

- The coordinate `x` is horizontal and `y` is vertical.

- $$ib = \begin{cases} 0 & \text{inside } \Omega \\ > 0 & \text{on } \partial\Omega \end{cases}$$

On $\partial\Omega$ it is equal to the boundary identification number.

The usual `if... then ... else` statement can be used with an important restriction on the logical expression which must return a **scalar** value:

```
1 if( logical  expression) then
2 {
3     statement;
4     ....;
5     statement;
6 }
7 else
8 {
9     .....
10 };
```

The `logical expression` controls the `if` by its return being 0 or $\neq 0$, it is evaluated only once (i.e. with `x, y` being the coordinates of the first vertex, if there are functions inside the logical expression). Auxiliary variables can be used.

In order to minimize the memory the symbol `:=` tells the compiler not to allocate a data array to this variable. Thus `v=sin(a*pi*x)`; generates an array for `v` but no array is assigned to `a` in the statement `a:=2` .

3.4.3 Value of a function at one point

If `f` has been defined earlier then it is possible to write `a:=f(1.2,3.1)`; Then `a` has the value of `f` at `x=1.2` and `y=3.1` .

It is also allowed to do

```
1 x1:=1.2;
2 y1:=1.6;
3 a:=f(x1,2*y1);
4 save('a.dta',a);
```


Remark: Recall that ,a being a scalar, its value is **appended** to the file `a.dta`.

3.4.4 Special functions: `dx()`, `dy()`, `convect()`

`dx(f)` is the partial derivative of `f` with respect to `x` ; the result is piecewise constant when `precise` is set and interpolated with mass lumping as a the piecewise linear function when `precise` is not set.

Note that `dx()` is a non local operator so statements like `f=dx(f)` would give the wrong answer because the new value for `f` is place before the end of the use of the old one.

The Finite Element Method does not handle convection terms properly when they dominate the viscous terms: upwinding is necessary; `convect` provides a tool for Lagrangian upwinding. By `g=convect(f,u,v,dt)` `Gfem` construct an approximation of

$$f(x - u1(x, y)dt, y - u2(x, y)dt)$$

Recall that when

$$\frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} + v \frac{\partial f}{\partial y} = \lim_{dt \rightarrow 0} \frac{f(x, y, t) - f(x - u(x, y)dt, y - v(x, y)dt, t - dt)}{dt}$$

Thus to solve

$$\frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} + v \frac{\partial f}{\partial y} - \text{div}(\mu \text{grad} f) = g,$$

in a much more stable way that if the operator `dx(.)` and `dy(.)` were use, the following scheme can be used:

```

1 iter(n) begin
2   id(f)/dt - laplace(f)*mu =g + convect(oldf,u,v,dt)/dt;
3   oldf = f
4 end;
```

Remark: Note that `convect` is a nonlocal operator. The statement `f = convect(f,u,v,dt)` would give an incorrect result because it modifies `f` at some vertex where the old value is still needed later. It is necessary to do

```

1 g=convect(f,u,v,dt);
2 f=g;
```

3.5 Global operators

It is important to understand the difference between a global operator such as `dx()` and a local operator such as `sin()` .

To compute `dx(f)` at vertex `q` we need `f` at all neighbors of `q`. Therefore evaluation of `dx(2*f)` require the computation of `2*f` at all neighbor vertices of `q` before applying `dx()` ; but in which memory would the result be stored? Therefore Gfem does not allow this and forces the user to declare a function `g =2*f` before evaluation of `dx(g)` ; Hence in

```
1 g = 2*f;  
2 h = dx(g) * dy(f);
```

the equal sign forces the evaluation of `g` at all vertices, then when the second equal signs forces the evaluation of the expression on the right of `h` at all vertices , everything is ready for it.

Global operators are

```
1 dx(), dy(), convect(), intt[], int()[]
```

Example of forbidden expressions:

```
1 intt[f+g], dx(dy(g)), dx(sin(f)), convect(2*u...)
```

3.6 Integrals

- **effect:**

`intt` returns a complex or real number, an integral with respect to `x,y` `int` returns a complex or real number, an integral on a curve

- **syntax:**

`intt[f]` or `intt(n1)[f]` or `intt(n1,n2)[f]` or `intt(n1,n2,n3)[f]`
`int(n1)[f]` or `int(n1,n2)[f]` or `int(n1,n2,n3)[f]`

where `n1,n2,n3` are boundary or subdomain identification numbers and where `f` is an array function.

```
1 border(1)... end; /* a border has number 1 */  
2 ... buildmesh(...);  
3
```

```

4 f = 2 * x;
5
6 /*
7  * nx,ny are the components of the boundary normal
8  */
9 g = f * (nx + ny);
10
11 /*
12  * can't do r:= int[2*x]
13  */
14 r:= int[f];
15 s:=int(1)[g];
16
17 /*
18  * this is the only way to display the result
19  */
20 save('r.dta',r);
21 save('s.dta',s);

```

- **Restrictions:**

`int` and `intt` are global operators, so the values of the integrands are needed at all vertices at once, therefore you can't put an expression for the integrand, it must be a function.

Be careful to check that the region number are correct when you use `intt(n)[f]` .

Unfortunately **freefem** does not store the edges numbers. Hence there are ambiguities at vertices which are at the intersections of 2 boundaries. The following convention is used: `int(n)[g]` computes the integral of `g` on all segments of the boundary (both ends have id boundary number !=0) with one vertex boundary id number = `n`. (Remember that you can control the boundary id number of the boundary ends by the order in which you place the corresponding **border** call or by an extra argument in **border**)

3.7 Solving an equation

3.7.1 Onbdy()

Its purpose is to define a boundary condition for a Partial Differential Equation (PDE).

The general syntax is

<pre>1 onbdy(ib1, ib2,...) id(u)+<expression>*dnu(u) = g 2 onbdy(ib1, ib2,...) u = g</pre>

where **ib**'s are boundary identification numbers, **<expression>** is a generic expression and **g** a generic function.

The term **id(u)** may be absent as in **-dnu(u)=g** . **dnu(u)** represents the conormal of the PDE, i.e.

$$\pm \vec{\nabla} u . n$$

when the PDE operator is

$$a * u - \vec{\nabla} . (\pm \vec{\nabla} u)$$

3.7.2 Pde()

The syntax for pde is

<pre>1 pde(u) [+ -] op1(u) [* /] exp1 [+ -] op2(u) [* /] exp2 ... = exp3</pre>

It defines a partial differential equation with non constant coefficients where **op** is one of the operator:

- **id()**
- **dx()**
- **dy()**
- **laplace()**
- **dxx()**
- **dxy()**
- **dyx()**
- **dyy()**

and where `[*/]` means either a `*` or a `/` and similarly for \pm . Note that the expressions are necessarily **AFTER** the operator while in practice they are between the 2 differentiations for `laplace...dyy`. Thus `laplace(u)*(x+y)` means

$$\nabla \cdot ((x+y) \nabla u)$$

.Similarly `dxy(u)*f` means

$$\frac{\partial f}{\partial x} \frac{\partial u}{\partial y}.$$

3.7.3 Solve()

The syntax for a single unknown function `u` solution of a PDE is

```

1 solve(u)
2 begin
3   onbdy()...;
4   onbdy()...;
5   ...;
6   pde(u)...
7 end;
```

For 2-systems and the use of `solve(u,v)`, see the section **2-Systems**. It defines a PDE and its boundary conditions. It will be solved by the Finite Element Method of degree 1 on triangles and a Gauss factorization.

Once the matrix is built and factorized `solve` may be called again by `solve(u,-1)...`; then the matrix is not rebuilt nor factorized and only a solution of the linear system is performed by an up and a down sweep in the Gauss algorithm only. This saves a lot of CPU time whenever possible. Several matrices can be stored and used simultaneously, in which case the sequence is

```

1 solve(u,i)...;
2 ...
3 solve(u,-i)...;
```

where `i` is a scalar variable (not an array function).

However matrices must be constructed in the natural order: `i=1` first then `i=2....` after they can be re-used in any order. One can also re-use an old matrix with a new definition, as in

```

1 solve(u,i)...;
2 ...
```

```

3 solve(u,i)...;
4 solve(u,\pm i)...;

```

Notice that `solve(u)` is equivalent to `solve(u,1)` .

Remark: 2-Systems have their own matrices, so they do not count in the previous ordering.

3.7.4 2-Systems

Before going to systems make sure that your 2 pde's are indeed coupled and that no simple iteration loop will solve it, because 2-systems are significantly more computer intensive than scalar equations.

Systems with 2 unknowns can be solved by

```

1 solve(u,v)
2 begin
3   onbdy(...) ...dnu(u)...=.../* defines a bdy condition for u */
4   onbdy(...) u =...          /* defines a bdy conditions for v */
5   pde(u) ...                 /* defines PDE for u */
6   onbdy(...)<v=... or ...dnu(v)...> /* defines bdy conditions for v */
7   pde(v) ...                 /* defines PDE for u */
8 end;

```

The syntax for `solve` is the same as for scalar PDEs; so `solve(u,v,1)` is ok for instance. The equations above can be in any orders; several `onbdy()` can be used in case of multiple boundaries...

The syntax for `onbdy` is the same as in the scalar case; either Dirichlet or mixed-Neumann, but the later can have more than one `id()` and only one `dnu()` .

Dirichlet is treated as if it was mixed Neumann with a small coefficient. For instance `u=2` is replaced by `dnu(u)+1.e10*u=2.e10` , with quadrature at the vertices.

Conditions coupling `u,v` are allowed for mixed Neumann only, such as `id(u)+id(v)+dnu(v)=1`. (As said later this is an equation for `v`).

In `solve(u,v,i) begin .. end;` when `i>0` the linear system is built factorized and solved. When `i<0` , it is only solved; this is useful when only the right hand side in the boundary conditions or in the equations have change. When `i<0`, `i` refers to a linear system `i>0` of **SAME TYPE**, so that scalar systems and 2-systems have their own count.

Remark: `saveall('filename',u,v)` works also.

The syntax for `pde()` is the same as for the scalar case. Deciding which equation is an equation for `u` or `v` is important in the resolution of the linear system (which is done by Gauss block factorization) because some block may not be definite matrices if the equations are not well chosen.

- A boundary condition like `onbdy(...) ... dnu(u) ... = ...;` is a boundary condition associated to `u`, even if it contains `id(v)` .
- Obviously a boundary condition like `onbdy(...) u...=...;` is also associated with `u` (the syntax forbids any `v` -operator in this case).
- If `u` is the array function in a `pde(u)` then what follows is the PDE associated to `u` .

3.7.5 Boundary conditions at corners

Corners where boundary conditions change from Neumann to Dirichlet are ambiguous because Dirichlet conditions are assigned to vertices while Neumann conditions should be assigned to boundary edges; yet **Gfem** does not give access to edge numbers. Understanding how these are implemented helps overcome the difficulty.

All boundary conditions are converted to mixed Fourier/Robin conditions:

```
1 id(u) a + dnu(u) b = c;
```

For Dirichlet conditions `a` is set to `1.0e12` and `c` is multiplied by the same; for Neumann `a=0` . Thus Neumann condition is present even when there is Dirichlet but the later overrules the former because of the large penalty number. Functions `a,b,c` are piecewise linear continuous, or discontinuous if `precise` is set.

In case of Dirichlet-Neumann corner (with Dirichlet on one side and Neumann on the other) it is usually better to put a Dirichlet logic at the corner. But if fine precision is needed then the option `precise` can guarantee that the integral on the edge near the corner on the Neumann side is properly taken into account because then the corner has a Dirichlet value and a Neumann value by the fact that functions are discontinuous.

3.7.6 Weak formulation

The new keyword `varsolve` allows the user to enter PDEs in weak form. Syntax:

```

1 varsolve(<unknown function list>;<test function list>
2          ,<<int>>) <<instruction>> : <expression>>;

```

where

- <unknown function list> and
- <test function list> are one or two function names separated by a comma.
- <int> is a positive or negative integer
- instruction is one instruction or more if they are enclosed within begin end or {}
- <expression> is an expression returning a real or complex number

We have used the notation << >> whenever the entities can be omitted.

Examples

```

1 varsolve(u;w) /* Dirichlet problem -laplace(u) =x*y */
2 begin
3     onbdy(1) u = 0;
4     f = dx(u)*dx(w) + dy(u)*dy(w)
5     g = x*y;
6 end : intt[f] - intt[g,w];
7
8 varsolve(u;w,-1) /* same with prefactorized matrix */
9 begin
10    onbdy(1) u = 0;
11    f = dx(u)*dx(w) + dy(u)*dy(w)
12    g = x*y;
13 end : intt[f] - intt[g,w];
14
15 varsolve(u;w) /* Neuman problem u-laplace(u) = x*y */
16 begin
17    f = dx(u)*dx(w) + dy(u)*dy(w) -x*y;
18    g = x;
19 end : intt[f] + int[u,w] - int(1)[g,w];
20
21 varsolve(u,v;w,s) /* Lamé's equations */

```



```

22 begin
23     onbdy(1) u=0;
24     onbdy(1) v=0;
25     e11 = dx(u);
26     e22 = dy(v);
27     e12 = 0.5*(dx(v)+dy(u));
28     e21 = e12;
29     dive = e11 + e22;
30     s11w=2*(lambda*dive+2*mu*e11)*dx(w);
31     s22s=2*(lambda*dive+2*mu*e22)*dy(s);
32     s12s = 2*mu*e12*(dy(w)+dx(s));
33     s21w = s12s;
34     a = s11w+s22s+s12s+s21w +0.1*s;
35 end : intt[a];

```

How does it works The interpreter evaluates the expression after the ":" for each triangle and for each 3 vertices; if there is an instruction prior the ":" it is also evaluated similarly. Each evaluation is done with one of the unknown and one of the test functions being 1 at one vertices and zero at the 2 others. This will give an element of the contribution of the triangle to the linear system of the problem. The right hand side is constructed by having all unknowns equal to zero and one test function equal to one at one vertex. whenever integrals appear they are computed on the current triangle only.

Note that `varsolve` takes longer than `solve` because derivatives like `dx(u)` are evaluated 9 times instead of once.

3.8 Results

3.8.1 plot, savemesh, save, load, loadmesh

Within a program the keyword `plot(u)` will display `u`.

Instruction `save('filename',u)` will save the data array `u` on disk. If `u` is a scalar variable then the (single) value of `u` is appended to the file (this is useful for time dependent problems or any problem with iteration loop.).

Instruction `savemesh('filename')` will save the triangulation on disk.

Similarly for reading data with `load('filename',u)` and `loadmesh('filename')`.

The file must be in the default directory, else it won't be found. The file format is best seen by opening them with a text editor. For a data array `f` it is:

```

1 ns
2 f[0]
3 ....
4 f[ns-1]

```

(**ns** is the number of vertices)

If **f** is a constant, its single value is *appended* to the end of the file; this is useful for time dependent problems or any problem with iteration loop.

If **precise** is set still the function stored by **save** is interpolated on the vertices as the P^1 continuous function given by mass lumping (see above).

For triangulations the file format is (**nt** = number of triangles):

```

1 ns nt
2 q[0].x q[0].y ib[i]
3 ...
4 q[n-1].x q[n-1].y ib[n-1]
5 me[0][0] me[0][1] me[0][2] ngt[0]
6 ...
7 me[n-1][0] me[n-1][1] me[n-1][2] ngt[n-1]

```

Remark: **Gfem** uses the Fortran standard for **me**[] [] and numbers the vertices starting from number 1 instead of 0 as in the C-standard. Thus in C-programs one must use **me**[] []-1 .

Remark: Other formats are also recognized by **freefem** via their file name extensions for our own internal use we have defined **.amdba** and **.am_fmt**. You can do the same if your format is not ours.

```

1      loadmesh('mesh.amdba'); /* amdba format (Dassault aviation) */
2      loadmesh('mesh.am_fmt'); /* am_fmt format of MODULEF */

```

Remark: There is an optional arguments for the functions **load**, **save**, **loadmesh**, **savemesh**. This is the 2nd or 3rd argument of these functions. Here are the prototypes:

```

1 save(<filename>, <function name>
2      [, <variable counter: integer or converted to integer>])
3 load(<filename>, <function name>

```

```

4      [, <variable counter: integer or converted to integer>])
5 savemesh(<filename>[, <variable counter:
6             integer or converted to integer>])
7 loadmesh(<filename>[, <variable counter:
8             integer or converted to integer>])

```

As an example see `nsstepad.pde` which use this feature to save the mesh and the solution at each adaptation of the mesh. This special feature allows you to save or load a generic filename with a counter, the final filename is built like this '`<generic filename>-<counter>`'.

3.8.2 Saveall()

The purpose is to solve all the data for a PDE or a 2-system with only one instruction. It is meant for those who want to write their own solvers.

The syntax is:

```

1 saveall('file_name', var_name1,...)

```

The syntax is exactly the same as that of `solve(,)` except that the first parameter is the file name. The other parameters are used only to indicate to the interpreter which is/are the unknown function.

The file format for the scalar equation (`laplace` is decomposed on `nuxx`, `nuyy`)

```

1 u=p   if Dirichlet
2 c u+dnu(u)=g if Neumann
3 b u-dx(nuxx dx(u))-dx(nuxy dy(u))-dy(nuyx dx))-dy(nuyy dy(u))
4   + a1 dx(u) + a2 dy(u) =f

```

is that each line has all the values for `x,y` being a vertex: `f`, `g`, `p`, `b`, `c`, `a1`, `a2`, `nuxx`, `nuxy`, `nuyx`, `nuyy`.

The actual routine is in C++

```

1 int saveparam(fcts *param, triangulation* t, char *filename, int N)
2 {
3     int k, ns = t->np;
4     ofstream file(filename);
5     file<<ns<<"    "<<N<<endl;
6     for(k=0; k<ns; k++)

```

```

7  {
8      file << (param)->f[k]<<" " ; file<<" ";
9      file << (param)->g[k]<<" " ; file<<" ";
10     file << (param)->p[k]<<" " ; file<<" ";
11     file << (param)->b[k]<<" " ; file<<" ";
12     file << (param)->c[k]<<" " ; file<<" ";
13     file << (param)->a1[k]<<" " ; file<<" ";
14     file << (param)->a2[k]<<" " ; file<<" ";
15     file << (param)->nuxx[k]<<" " ; file<<" ";
16     file << (param)->nuxy[k]<<" " ; file<<" ";
17     file << (param)->nuyx[k]<<" " ; file<<" ";
18     file << (param)->nuyy[k]<<" " ; file<<" ";
19     file << endl;
20 }
21 }

```

The same function is used for complex coefficients, by overloading the operator `jj`:

```

1 friend ostream& operator<<(ostream& os, const complex& a)
2 {
3     os<<a.re<<" " << a.im<<" ";
4     return os;
5 }

```

For 2-systems also the same is used with

```

1 ostream& operator<<(ostream& os, cvect& a)
2 {
3     for(int i=0; i<N;i++)
4         os<<a[i]<<" ";
5     return os;
6 }
7 ostream& operator<<(ostream& os, cmat& a)
8 {
9     for(int i=0; i<N;i++)
10         for(int j=0; j<N;j++)
11             os<<a(i,j)<<" ";
12     return os;
13 }

```

where $N=2$.

A Dirichlet condition is applied whenever $p[k](?)$. (Dirichlet conditions with value 0 are changed to value $1e-10$)

3.9 Other features

Gfem supports other interesting features:

3.9.1 Iter

The syntax is:

```
1 iter(j){....}
```

where j refers to the number of loops; j can be the result of an expression (as in `iter(i*k)`).

Imbedded loops are not allowed. You can use `iter` with the adaptation features of **Gfem**.

3.9.2 Complex numbers

Gfem can handle complex coefficients with 4 dedicated keywords:

- **complex** : to tell Gfem that complex number will be used. When it is used it must be located at the beginning of the program before any function declarations, otherwise the results will be incorrect. It can appear more than once in the program but only the first occurrence counts.
- **I**: for `sqrt(-1)` .
- **Re** : for the real part.
- **Im** : for the imaginary part.

There is purposely no `conj` function but `barz=Re(z)-I*Im(z)` will do. By default all graphics display the real part. To display the imaginary part do `plot(Im(f))`.

The functions implemented for complex numbers are:

- *cos*
- *sin*

- z^x where z is complex and x is a float

The linear systems for the PDE are solved by a Gauss complex LU factorization.

WARNING: failure to declare `complex` in the program implies all computation will be done in real, even if `I` is used.

3.9.3 Scal()

The instruction `a:=scal(f,g);` does

$$a = \int_{\Omega} f(x,y)\bar{g}(x,y)dx dy$$

where Ω is the triangulated domain.

3.9.4 Wait, nowait, changewait

Whenever there is a `plot` command, **Gfem** stops to let the user see the result. By using `nowait` no stop will be made;

- `wait` turns back the stop option on.
- `changewait` toggles the option from on to off or off to on.

Remark under X11: If you click the right button in the window, the next time the solver will give the hand to the plotter the program will stop.

3.9.5 One Dimensional Plots

This function is **only available** under integrated environments.

The last function defined by the keyword `plot` is displayed. It can be visualized in several fashion, one of which being a **one dimensional plot** along any segment defined by the mouse. Selection of this menu brings causes Gfem to waits for the user input which should be the line segment on which the function is to be displayed. Thus one should **press the mouse at the beginning point then drag the mouse and release the button at the final point**

It is safe to click in the window after to check that the function display is correct. What is seen is a

$$t \rightarrow f(x(t), y(t))$$

Plot where $[x(t), y(t)]_t$ is the segment drawn by the user and **f** is the last function displayed in the plot window.

The abscissa is the distance with the beginning point.

3.9.6 Precise

This keyword warns **Gfem** that precise quadrature formula must be used. Hence array-functions are discretized as piecewise linear **discontinuous** functions on the triangulation. Then all integrals are computed with 3 inner Gauss points slightly inside each triangle.

This option consumes more memory (3nt instead of ns per functions, i.e. 9 times more approximately, but still it is nothing compared with the memory that a matrix of the linear system of a PDE requires) because each array-function is stored by 3 values on each triangles.

It is a good idea to use it in conjunction with **convect** and/or discontinuous nonlinear terms.

3.9.7 Exec(), user(), how to link an external function to Gfem

`exec('prog_name')` will launch the application `prog_name`. It is useful to execute an external PDE solver for instance especially under Unix. It is not implemented for Macintosh because there is no simple way to return to MacGfem after `programe` has ended. The same can be achieved manually by a suitable combination of `saveall`, `wait` and `load` and simultaneous execution under multifinder.

`user(what,f)` calls the C++ function in a j-loop:

```
1 for (int j=0; j<nquad, j++)
2   creal gfemuser( creal what, creal* f, int j)
```

- `creal` is a scalar (`float`) or a complex number if `complex` has been set; `nquad=3*nt` if `precise` is set and `ns` otherwise.
- `what` is intended for users who need several such functions. Then all can be put in the super function `user` and selection is by an if statement on `what`.
- Within `gfemuser` access to all global variables are of course possible: the triangulation (`ns, nt, me, q, ng, ngt, area...`) ... refer to the file `fem.C` for more details.

Remark: An example of such gfemuser function is in `fem.C` ; if you wish to put your own you must compile and link it. Under Unix, it is easy. Under the Macintosh system, either you use freefem, which is Gfem's kernel, or you must ask us a library version of MacGfem.

3.10 Language internals

- Like most interpreters it has a lexical and a syntactic part. In the lexical part the source is broken into tokens and recognized as symbols (see the enum symbol in the source file `lexical.h`). For instance if the first character is a digit then it is a number and the symbol type associated is `cste`. This job is done by function `nextsym`. In addition it constructs a table of constants and variables (which for convenience contains also the reserved words of the language).
- The lexical analyzer is a function called by the syntax analyzer. Hence the second is the main routine in the program except for a few initialization; its name is `instruction`. The syntax analysis is driven by the syntax rules because the language is LL(1). Thus there is C-function for each non-terminal.
- The program does not generate an object code but a tree. For example, parsing `x * y` would generate a tree with root `*` and two branches `x` and `y`. Trees are C-struct with four pointers for the 4 branches (here two would point to NULL) and a symbol for the operator. The C-function which builds trees is called `plante`. In the end the program is transformed into a full tree and to execute the program there is only one thing to do: evaluate the operator at the root of the tree.
- The evaluation of the program is done by the C-function `eval`. It looks at the root symbol and perform the corresponding operation. Here it would do: *return "value pointed by L1"*"value pointed by L2" if L1 and L2 where the addresses of the two branches.*
- The art of the game is to associate a tree to each operation. For example when the value of the variable `x` is required, this is also done by a tree which has the operator "oldvar" as root. The trickiest of all is the compound instruction `{...;...};`. Here `{` is considered as an operator with one branch on the current instruction and one branch on the next one. Similarly for the `if...then... else` instruction.

Suppose one wants to add an instruction to **freefem**, here is what must be done:

- Make sure the syntax is LL(1) and does not conflict with the old one.
- Add reserved words to the table with **installe** . As there will be a new Symbol, update the list of symbols (an enum structure). Add the C-functions for the syntax analysis according to the diagrams. Modify eval by adding to the **switch** the new case.

Here is the very simple structure we used for the nodes of the tree:

```
1 typedef struct noeud
2 {
3     Symbol symb;
4     creal value;
5     ident *name;
6     long  junk;
7     char  *path;
8     struct noeud *l1, *l2, *l3, *l4;
9 } noeud, *arbre;
```

Chapter 4

Examples

4.1 Triangulations examples

- **A UNIT RING (INNER RADIUS IS 0.25)**

```
1 twopi := 2*pi;  
2 border(1,0,twopi,60) { x := cos(t); y := sin(t) };  
3 border(2,0,twopi,20) { x := 0.25*cos(-t); y := 0.25*sin(-t) };  
4 buildmesh(400);
```

- **THE RECTANGLE [(0,0),(0,2),(2,1),(0,10)]**

```
1 border(1,0,2,20) { x:= t; y:= 0 };  
2 border(1,0,1,10) { x:= 1; y:= t };  
3 border(1,0,2,20) { x:= 2-t; y:= 1 };  
4 border(1,0,1,10) { x:= 0; y:= 1-t };
```

- **A SQUARE WITH WELL IDENTIFIED SIDES AND CONTROL OF IB AT CORNERS**

```
1 border(1,0,4,41)  
2 {  
3   if(t<=1) then { x:=t; y:=0 };  
4   if((t>1)and(t<2)) then { x:=1; y:=t-1; ib:= 2 };  
5   if((t>=2)and(t<=3)) then { x:=3-t; y:=1; ib:= 3 };  
6   if(t>3) then { x:=0; y:=4-t; ib:= 4 }  
7 };  
8 buildmesh(400);
```

- **MULTI-REGIONS CIRCLE**

```

1 border(1,0,2,17) {x:= cos(pi*t); y:= sin(pi*t)};
2 border(0,-1,1,7) { x:= t; y:=0; };
3 border(0,0,1,4) { x:=0;y:=t };
4 buildmesh(300);
5 /* observe the value of "region" by using "show triangle numbers */

```

4.2 Scalar examples

- **ELECTROSTATIC CONDENSOR**

```

1 /* a circle of radius 5 centered at (0,0) */
2 border(1,0,2*pi,60) begin x := 5 * cos(t); y := 5 * sin(t) end ;
3 /* The rectangle on the right */
4 border(2,0,1,4) begin x:=1+t; y:=3 end ;
5 border(2,0,1,24) begin x:=2; y:=3-6*t end ;
6 border(2,0,1,4) begin x:=2-t; y:=-3 end ;
7 border(2,0,1,24) begin x:=1; y:=-3+6*t end ;
8 /* The rectangle on the left */
9 border(3,0,1,4) begin x:=-2+t; y:=3 end ;
10 border(3,0,1,24) begin x:=-1; y:=3-6*t end ;
11 border(3,0,1,4) begin x:=-1-t; y:=-3 end ;
12 border(3,0,1,24) begin x:=-2; y:=-3+6*t end ;
13 buildmesh(800);
14
15 /* Boundary conditions and PDE */
16 solve(v)
17 begin
18   onbdy(1) v = 0;
19   onbdy(2) v = 1;
20   onbdy(3) v = -1;
21   pde(v) -laplace(v) =0;
22 end;
23 plot(v);

```

- **HEAT CONDUCTION AND RADIATION**

```

1 border(1,0,22,89)
2 begin

```

```

3  if(t<=10)then begin x:= t; y:=0 ; ib:=3 end;
4  if((t>10)and(t<11))then begin x:=10; y:=t-10; ib:=2 end;
5  if((t>=11)and(t<=21))then begin x:=21-t; y:=1; ib:=4 end;
6  if(t>21)then begin x:=0; y:=22-t end;
7  end;
8  buildmesh(800);
9
10 changewait;
11 t0 := 10; t1 := 100; te := 25; b=0.1; c = 5.0e-8;
12 w = (b + 2*c * (te+546)*(te+273)*(te+273));
13 solve(v,1)
14 begin
15   onbdy(1) v=t0; onbdy(2) v = t1; onbdy(3) dnu(v)=0;
16   onbdy(4) id(v) * w + dnu(v) = te * w;
17   pde(v) -laplace(v) * y =0;
18 end;
19 iter(10)
20 begin u=v;
21   w = (b + c * (u+te + 546)*((u+273)*(u+273) + (te+273)*(te+273)));
22 solve(v,-1) begin
23   onbdy(1) v=t0; onbdy(2) v = t1;
24   onbdy(3) dnu(v)=0; onbdy(4) id(v)*w + dnu(v)= te * w;
25   pde(v) -laplace(v) * y =0; plot(v);
26 end;
27 end

```

• HEAT: NON HOMOGENEOUS MATERIAL

```

1  r0 := 1.0; r1 := 2.0;
2  border(1,0,22,89)
3  begin
4    region :=1;
5    if(t<10)then begin x:= t; y:=0 ; ib:=3 end;
6    if((t>=10)and(t<=11))then begin x:=10; y:=r1*(t-10); ib:=2 end;
7    if((t>11)and(t<21))then begin x:=21-t; y:=r1; ib:=4 end;
8    if(t>=21)then begin x:=0; y:=r1*(22-t) end;
9  end;
10 border(0,0,10,41) begin x:= t; y:=r0 end;
11 buildmesh(800);
12

```

```

13 t0 = 10; t1 = 100; te := 25; kappa = 0.01 + max(y-1,0)/(y-1.0001);
14 solve(v)
15 begin
16 onbdy(1) v=t0;
17 onbdy(2) v = t1;
18 onbdy(4) dnu(v)=0.2;
19 onbdy(3) dnu(v)=0;
20 pde(v) -laplace(v)*kappa*y +id(v)*kappa*y =0;
21 plot(v);
22 end;

```

• COMPRESSIBLE POTENTIAL FLOW

```

1 changewait; /* gamma = 1.4, outer circle radius is 5 */
2 mach1 := 1/sqrt(6); machinfy = 0.85*mach1;
3 rhoinfy=sqrt((1-machinfy^2)^5);
4 solve(phi) begin
5 onbdy(1) dnu(phi) = rhoinfy*machinfy*x/5; onbdy(2) dnu(phi) = 0;
6 pde(phi) id(phi)*0.0001-laplace(phi) = 0;
7 end;
8 u1 = dx(phi); u2 = dy(phi); rho=sqrt((1-(u1^2+ u2^2))^5); plot(phi);
9
10 iter(5)
11 begin
12 solve(phi)
13 onbdy(1) dnu(phi) =rhoinfy*machinfy*x/5; onbdy(2) dnu(phi) = 0;
14 pde(phi) id(phi)*0.0001-laplace(phi)*rho=0;
15 end;
16 u1 = dx(phi); u2 = dy(phi); rho=sqrt((1-(u1^2+ u2^2))^5);
17 rhop = convect(rho,u1,u2,0.1); plot(rho)
18 end;
19 plot(sqrt((u1^2+u2^2))/mach1);

```

• NAVIER STOKES EQUATIONS

```

1 /* Poor but better than none algorithm*/
2 changewait;
3 border(1,0,1,6) begin x:=0; y:=1-t end;
4 border(2,0,1,15) begin x:=2*t; y:=0 end;
5 border(2,0,1,10) begin x:=2; y:=-t end;

```

```

6 border(2,0,1,20) begin x:=2+3*t; y:=-1 end;
7 border(2,0,1,35) begin x:=5+15*t; y:=-1 end;
8 border(3,0,1,10) begin x:=20; y:=-1+2*t end;
9 border(4,0,1,35) begin x:=5+15*(1-t); y:=1 end;
10 border(4,0,1,40) begin x:=5*(1-t); y:=1 end;
11 buildmesh(900);
12
13 nu = 0.002; dt := 0.4;
14
15 /* initial pressure */
16 solve(p,1)
17 onbdy(1)dnu(p) = -2*nu;
18 onbdy(3) p=0; onbdy(2,4) dnu(p) = 0;
19 pde(p) - laplace(p)= 0;
20 end;
21 /* initial horizontal velocity */
22 solve(u,2) begin
23 onbdy(1) u = y*(1-y);
24 onbdy(3) dnu(u) = 0; onbdy(2,4) u = 0;
25 pde(u) id(u)/dt-laplace(u)*nu = -min(y*y-y,0)/dt;
26 end;
27 /* initial vertical velocity */
28 solve(v,3)begin
29 onbdy(1,3)v = 0; onbdy(2,4) v = 0;
30 pde(v) id(v)/dt-laplace(v)*nu =0;
31 end;
32 un = u; vn = v;
33 iter(80)
34 begin f=convect(un,u,v,dt); g=convect(vn,u,v,dt);
35 /*Horizontal velocity*/
36 solve(u,-2) begin
37 onbdy(1) u = y*(1-y); onbdy(2,4) u = 0;
38 onbdy(3)dnu(u)=0;
39 pde(u) id(u)/dt-laplace(u)*nu = f/dt -dx(p);
40 end;
41 plot(u);
42 /* Vertical velocity */
43 solve(v,-3) begin
44 onbdy(1,2,3,4) v = 0;
45 pde(v) id(v)/dt-laplace(v)*nu = g/dt -dy(p);

```

```

46     end;
47 /* Pressure */
48     solve(p,-1) begin
49         onbdy(1)dnu(p) =-2*nu;
50         onbdy(3) p=0; onbdy(2,4) dnu(p) = 0;
51         pde(p) -laplace(p)= -(dx(f) + dy(g))/dt;
52     end;
53     un = u; vn = v;
54 end ;
55     save('u.dta',u); save('v.dta',v); save('p.dta',p); plot(u);

```

4.3 Complex number example

```

1 complex; nowait;
2 border(1,0,1,10) begin x:=t; y:=0; end;
3 border(1,0,1,10) begin x:=1; y:=t; end;
4 border(2,0,1,10) begin x:=1-t; y:=1; end;
5 border(1,0,1,10) begin x:=0; y:=1-t; end;
6 buildmesh(200);
7
8 solve(u) /* observe than Re(u) = Im(u) */
9 begin
10     onbdy(1,2) u=0;
11     pde(u) id(u)-laplace(u)=1+I ;
12 end;
13 v=Im(u);
14 plot(u);plot(v);plot(u-v);

```

4.4 2-system example

```

1 /* This is a 2-system example for which the solution is know
2 analytically, thus the precision of Gfem can be checked */
3 nowait;
4 ns:=40;
5 border(1,0,2*pi,2*ns) begin x:= 3*cos(t); y:= 2*sin(t); end;
6 border(2,0,2*pi,ns)    begin x:= cos(-t); y:= sin(-t); end;
7 buildmesh(ns*ns);
8

```

```

9 ue= sin(x+y);
10 ve = ue;
11 p = ue;
12 nx = -x;
13 ny =- y;
14 dxue = cos(x+y);
15
16 c = 0.2;
17 a1 = y;
18 a2 =x;
19 nu = 1;
20 nu11 = 1;
21 nu22 = 2;
22 nu21 =0.3;
23 nu12 =0.4;
24 b=1;
25
26 dnuue=dxue*(nu*(nx+ny) +
27 (nu11 + nu12)*nx + (nu21+ nu22)*ny);
28 g = ue*c+dnuue;
29 f = b*ue+dxue*(a1+a2) +ue*(2*nu+nu11+nu12+nu21+nu22);
30
31 solve(u,v) begin
32 onbdy(1) u = p;
33 onbdy(1) v = p;
34 onbdy(2)
35     id(u)*c/2 + id(v)*c/2 + dnu(u) = g;
36 onbdy(2) id(v)*c + dnu(v) = g;
37 pde(u) id(u)*b + dx(u)*a1 + dy(u)*a2
38     -laplace(u)*nu - dxx(u)*nu11 -
39     dxy(u)*nu12 - dyx(u)*nu21 - dyy(u)*nu22 =f;
40
41 pde(v) id(v)*b/2+id(u)*b/2
42     + dx(v)*a1 + dy(v)*a2 -laplace(v)*nu
43     - dxx(v)*nu11 - dxy(v)*nu12 -
44     dyx(v)*nu21 - dyy(v)*nu22 =f;
45 end;
46
47 plot(abs(u-ue) + abs(v-ve));

```


Chapter 5

GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 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 written 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. Secondly, 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.

5.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

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. (For example, 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.

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 “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

5.2 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.

5.3 Copying in Quantity

If you publish printed copies 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 publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. 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.

5.4 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:

- 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.
- 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 less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.

- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- 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.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled “History”, and 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.
- 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.
- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- 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.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

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.

5.5 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.

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.”

5.6 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.

5.7 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, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they 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 quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

5.8 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 provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

5.9 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.

5.10 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 © YEAR YOUR NAME. Permission is granted to
copy, distribute and/or modify this document under the terms of

the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

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

Index

array-function, 3

border, 9

buildmesh, 9, 11

complex, 27

convect, 13, 15, 29

diagonal swaping, 11

domain decomposition, 11

dx, 15

dy, 15

exec, 29

http, 4

iter, 27

loadmesh, 23, 24

Mesh formats

- ambda, 24
- gfem, 24
- am_fmt, 24

polygon, 9, 11

precise, 29

project, 3

savemesh, 23, 24

scal, 28

solve, 19, 20, 25

triangulation, 3

user, 29

vertex, 3