FREEFEM+
for Macs, PCs, Linux

ftp://ftp.ann.jussieu.fr/pub/soft/pironneau

D. Bernardi, F. Hecht, K Ohtsuka, O. Pironneau
Dominique.Bernardi@math.jussieu.fr,
Frederic.Hecht@inria.fr,
Olivier.Pironneau@ann.jussieu.fr
ohtsuka@barnard.c.hiroshima-dit.ac.jp

December 25, 1998

Contents

1 Generalities

1.1 Contacts

- Olivier.Pironneau@inria.fr,
- http://www.asci.fr
- ftp://ftp.ann.jussieu.fr/pub/soft/pironneau/

If you wish to know more about the Finite Element Method used in freefem you may consult

"B. Lucquin, O. Pironneau:*Scientific Computing for Engineers* Wiley 1998."

For automatic mesh generation, you may consult

"P.L. George: *Automatic triangulation*, Wiley 1996"

and the documentation of "bang"

"F. Hecht: The mesh adapting software; bang. INRIA report 1998."

1.2 Source codes

These web sites contain the source files, examples and exec files for Macintosh PPC, Windows PC and Linux Unix machines.
Some makefiles are given too. Recompilation of source files are easy on

- MetroWerks CodeWarrior C++compiler (Macs & PCs) version >3.0. It should be compiled as a "SIOUX C++ application" or a "SIOUX C++ Win32 appl" (with 16 bits alignment in link option for Windows).
- gnu gcc 2.7 or greater on unixx system.

We have made extensive use of templates so other compilers may work if they conform to the latest standard of the C++ language (which is not the case of Visual C++ for instance).

1.3 Legal conditions

1.3.1 Warning

freefem is a scientific product to help you solve Partial Differential Equations in 2 dimensions; it assumes a basic knowledge and understanding of the Finite Element Method and of the Operating System used. It is also necessary to read carefully this documentation to understand the possibilities and limitations of this product. the authors are not responsible for any errors or damage due to wrong results.

1.3.2 Freeware

freefem is a freeware. It is handed out on an "as is" basis. The authors exclude any and all implied warranties, including warranties of merchantabilities and fitness for a particular purpose. Anyone is allowed to use it, except for commercial purpose within another software. It is illegal to use part of the source code to develop another product without the authorization of the authors. The banner in each source code file cannot be removed from any distribuable copy.

2 Overview

Freefem is a user friendly software for solving systems of Partial Differential Equations (PDEs) in two dimensions. It was released in 1995 by the same authors together with C. Prud'homme and P. Parole as a followup of MacFEM. The current version is freefem 3.0. It is based on a finite element solver, mesh adaption was introduced later in the software after completion by M. Castro's thesis. MacGfem, PCGfem, XGfem are commercial WYSIWYG encapsulation of freefem.
But mesh adaption became an important feature of freefem; originally written in C we made some changes in C++ but the software became difficult to manage.

freefem+ is a proper implementation of the same idea entirely rewritten in in C++. The general philosophy of freefem is kept but there are a number of new features which required some modification to the syntax of freefem. We give here an overview of these. The great power of freefem+ over freefem is:

- It can handle multiple meshes within one program
- It has an extremely powerful interpolator from one mesh to another
- It has a very robust an versatile mesh adaption module
- It can handle systemsof PDEs both in strong and weak forms

The drawback is that this piece of software is still a dynamite stick and it blast on the user more often than not because the number of "errors" due to a misunderstanding of freefem is large... very large! So if you like racing cars go on with freefem+, otherwise stick to freefem 3.0 and drive the VW.

To use freefem+ you should open your favorite text editor, type a program, name the file "something.edp" then execute freefem+ on that file. Minimal graphics are displaid within freefem+ and more elaborate ones can be obtained with other packages from files generated by freefem (gnuplot in particular). We decided to concentrate on the applied math part rather than on the interface, with the idea that everything should be possible (postscript out of drawings especially).

2.1 Meshes

Mesh adaption is not efficient unless the grid points are allowed to slide on the boundaries for optimal repartition. Hence borders are curves which now exists by they analyical definition throughout the program:

```
border c(t=0,2*pi){x = cos(t); y = sin(t)};
mesh mesh0 = buildmesh(c(25));
mesh mesh1 = buildmesh(c(50));
print("pi+2pi=", int(mesh0)(1) + int(c)(1));
```

This program creates two triangulations of the unit disk, one with 25 points on the boundaryanother with 50. Then it prints the sum of the area of the unit disk with its circumference.

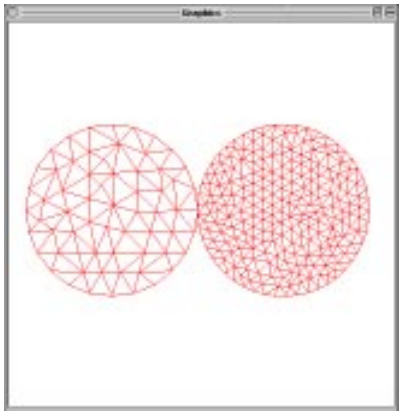

Figure 1
This plot is a screen shot obtain by adding the two lines

```
mesh mesh2=movemesh(mesh1, x+2,y);
plot(mesh0,0,mesh2,0);
```

2.1.1 interpolation

Mesh adaption means that a finite element function $u(x, y)$ is no longer defined by an array of values on the vertices of a mesh but exists independantly of the mesh on which it was created. Interpolation becomes a real issue. Freefem+ provides a fast and efficient way to interpolate a function on any mesh, it uses quadtrees, background meshes, virtual triangles... the result is fast! You should not notice any difference in speed between the two plots:

```
array(mesh1) u = x^2 + y^2;
plot(mesh1,u); // plot u on its own mesh
plot(mesh0,u); // plot u interpolated on another mesh
```

How about this one:

```
array(mesh0) v = sin(x) + sin(y);
plot(u + v); // u is defined on mesh1 and v on mesh0
```

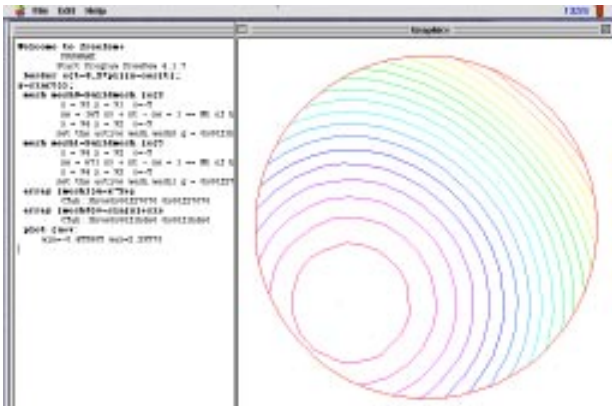

Figure 2

This screen shot shows the result and the entire Macintosh screen with the text window which displays in bold the line of the freefem program executed. Here the mesh is not prescribed for the plot, so the *active mesh* is used; it is the last one appearing after the keyword "mesh", here mesh1 (in the first example). Outside its domain of definition the interpolator assign the value equal to that of the nearest boundary point. Most commands accept default mesh specification.

2.1.2 Operations on Meshes

Meshes can be

- Created, by the keyword *buildmesh*.
- Moved or deformed, by the keyword *movemesh*.
- Read from disk by *readmesh*
- Saved on disk by *savemesh* in several different formats specified by the suffix part or the file name.
- adapted by using the keyword *adaptmesh*

Examples

```
/* Example 1 */
mesh th = readmesh("amesh.msh"); // the ".msh" determines the format
read("afunction.dat",u);
plot(th,u);
mesh sh = adaptmesh(th,u);
savemesh("unicemesh.msh");

/*  Example 2 */
border a(t=0,2*pi){x = 0.25*cos(t); y = 0.125*sin(t)};
mesh th = buildmesh(a(-15) + c(30)) // unit circle with hole
mesh sh = movemesh(th,x+y,x-y);// apply to th map X=x+y, Y=x-y
mesh th = adaptmesh (th,u); // adapt the mesh to min err on u
mesh sh;  // now sh is the default active mesh.
```

Notice that meshes can be reused as in the next to last line. Notice also that the hole is obtain by specifying a boundary with a minus sign inside, meaning that the boundary is scanned counterclockwise. By definition domains are on the left side of their oriented boundaries.

2.1.3 One more example

Polygons or shapes best described by intersections of curves, like the unit square are entered like this

```
border a(t=0,1){x=t; y=0};
border b(t=0,1){x=1; y=t};
border c(t=1,0){x=t; y=1};
border d(t=1,0){x=0; y=t};
n:=20;
mesh th= buildmesh(a(n)+b(n)+c(n)+d(n));
```

(notice that the number of points pr boundary is an "expression", here "n"). But boundaries can cross at vertices only.
Another way to enter he unit square is

```
border a(t = 0,4){
                 if(t<=1)then{x=t;   y=0  }
else  if((t>1)*(t<=2))  then{x=1;   y=t-1}
```

```
else  if((t>2)*(t<=3))  then{x=3-t; y=1  }
                        else{x=0;   y=4-t}
};
mesh th= buildmesh(a(81));
```

There may be an advantage in having a single name (here "a") for the entire boundary, but a good triangulation is harder to get this way. To group the boundary a,b,c,d of the first method into one unit look at the instruction "R2(x,y,ref)" and the use of "ref".

2.2 Data Types

Already in freefem there was a difference between numbers and arrays (piecewise linear continuous functions on a mesh are fully specified from the array of its values at the vertices). In freefem+ there are 3 types of data:

```
number r = 0.01;                 // a number occupies one memory
array(th0) u = x+ exp(y);      // an array occupies nv memories
function f(x,y,z) = z * x + r * log(y);// this is a definition
```

The first has one value stored in one memory location;
The second has one value per vertex of the active mesh,and these are computed at execution time when the instruction "array"..." is encountered, the definition of "u" is then lost thereafter.
The last line is actually a definition and it is used only when f is used in an instruction which is eveluated. Thus if we write

```
append("test.dta",u(0.1,2.3) + f(0.4,5.6,2));
```

we will see 0.1 + P(exp(2.3)) + 2*0.4 + 0.01*log(5.6) printed on the screen and written at the end of file "test.dta" (append is print+save)
Here P is the linear interpolation operator on the mesh from the values at the vertices because u(0.1,2.3) means u(x,y) at these values : arrays are implicit functions of the two coordinates.
For f(0.4,5.6,2), the expression that defines f above is simply evaluated with the parameters x=0.4, y=5.6, z=2. The mechanism to compute u(0.1,2.3) is entirely different, a is first computed at all vertices and then an interpolation is done when non vertex values for x,y are used.

2.2.1 Nearer to Math but "racing car" Notations

We decided not to force type declaration for a data because it makes the program harder to read. A short form of the above is

```
r:= 0.01;
array u = x+ exp(y);
f(x,y,z) = z * x + r * log(y);
```

The default type is the function; to get a number one must use the := or the "number" qualifyer; to get an array one must explicitly use the qualifyer "array" the first time the variable is introduced. Arrays and numbers can be reuseable but not functions. This means the following is valid

```
r := 0.02; // redefines r, OK
array u = x+ sin(y); // redefines u, OK
g(z) = z * x + r * log(y);// same as f above
h = x+log(y); // short for h(x,y) = x+log(y);
f = x-log(y); // not allowed, redefines f
```

The last line will produce a compile time error because it redefines "f".

2.3 Partial Differential Equations

To input a PDE or a system of PDE one must use either its strong form or its weak form. Here is an example of the use of a strong form of

$$w - 2\Delta w = f, \qquad 2\frac{\partial u}{\partial n} + w = 1$$

```
border a(t=0,1){x=t; y=0};
border b(t=0,1){x=1; y=t};
border c(t=1,0){x=t; y=1};
border d(t=1,0){x=0; y=t};
mesh th= buildmesh(a(20)+b(20)+c(20)+d(20));

r:= 0.1;
array u = x+y;
f(x,y,z) = z * x + r * log(1+y);
plot(f(x,y,0.3));
```

```
solve(th,w) {
     pde(w) w - laplace(w) * 2 = f(x,y,0.3);
     on(c) w = u;
     on(b) dnu(w) + w = 1
};

plot(w);
```

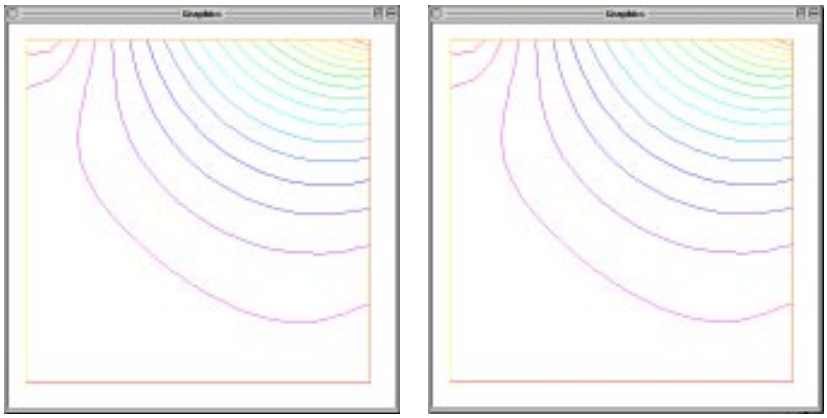

Figure 3

The results are shown on the left graphic above (the right one is the same problem solved by the weak form, the small differences are due to the different quadrature formulae).
Notice that on boundaries "a" and "d" we haven't specified any condition; this means that it is homogeneous Neumann. Indeed, freefem builds the weak form from the data, discretizes it by the Finite Element Method of degree 1 on triangles and solves the discrete linear system by an LU Gauss factorization.
This is why "dnu" is not the normal derivative but the conormal derivative, here twice the normal, i.e. for the PDE

$$\alpha u + v \cdot \nabla u - \nabla \cdot (A \nabla u) = f$$

the conormal is $A\ n$ if n is the outer normal. This, by the way is the most general linear scalar second order PDE and freefem+ can solve it.
The same example in weak form would be

```
varsolve(th) aa(w,W)
   with aa = int() (w*W + (dx(w)*dx(W)
                    + dy(w)*dy(W))*2 - f(x,y,0.3)*W )
             + on(c)(W)(w =u)
             + int(b) (w*W-W) ;
```

freefem+ also handles systems (here Lam's problem)

```
solve(u,v) {
     pde(u) - laplace(u)*mu - dxx(u) - dxy(v) = f1;
     on(c) u = 0;
     pde(v) - laplace(v)*mu - dyy(v) - dyx(u) = f2;
     on(b) dnu(u) + v = 1;
     on(c) v = 0;
};
```

The general syntax is to write for each PDE the boundary conditions which are naturally associated with it then the next PDE...
The Stokes system in variational form is

```
varsolve aa(u,u1,v,v1,p,p1) with
   aa = int()(dx(u)*dx(u1)+dy(u)*dy(u1)
        + dx(v)*dx(v1)+dy(v)*dy(v1) + dx(p)*u1+dy(p)*v1
        -dx(p1)*u - dy(p1)*v + p*p1)
        + on(b)(u1)(u = 0)+ on(c)(u1)(u = 1)
        + on(b,c)(v1)(v = 0);
```

All unknowns are followed by their corresponding test functions in the declaration "varsolve"; here u1 is the test function corresponding to u... Dirichlet conditions are declared by the keyword "on" and the test function following indicates the position of that equation in the linear system generated. Linear sytems are solved by Gauss LU factorization and hence can be re-used such as in

```
dt:=0.1;
```

```
array(th1) U = tan(x+y);
for j=0 to 5 do{
    solve(U) with C(j) { pde(U) U/dt - laplace(U) = U/dt;
                         on(b,c) U=0; };
};
```

A loop is a powerful element of freefem+ because it allows time dependant and/or coupled systems. Above is an example of Euler scheme in time for the heat equation

$$\partial_t u - \Delta u = 0, \quad u|_{b,c} = 0$$

where the matrix C is factorized the first time (j=0) and then reused after ($j \neq 0$).

2.3.1 Implementation strategy

Notice that the integrals involves products of functions on the two meshes. When this happens the quadrature points are on the midedges of the active triangulation. Hence some control on these is given by specifying a mesh name in the parameter list of "int". For instance

```
varsolve(SH,i) AA(U,W) with {
   AA = int(sh)( (U-Uold)*W + dx(U)*dx(W)+dy(U)*dy(W)-W
    + dx(uold)*dx(W)+dy(uold)*dy(W) ) + on(e,e1)(W)(U=0);
```

would force to use for quadrature points the mid-edges of the triangulation sh. To do that efficiently we store the list of triangles of sh which intersect each triangles of SH and the assemblage of the matrix AA is done by

```
Set AA(i,j) = 0 for all i,j in [0,nv(SH)]
Create 2 arrays U and W of size=nb of vertices of SH
for each triangle Tk in SH do
   loop on the 3 vertices  q(i) of Tk,
     loop on the 3 vertices q(j) of Tk,
      With W(i) = 1, U(j) = 1 and 0 on all others
      compute Ck = int(Tk)(U*W+dx(U)*dx(W)+dy(U)*dy(W))
       by using the quadrature points of sh.
       AA(i,j) = AA(i,j) + Ck
     end loop i
```

```
    end loop  j
  end loop k
```

The right hand side of the linear system is computed likewise and the Dirichlet condition is a postprocessing of AA by penalty.
So it must be noted that when the triangulation argument in "int()" is different from the triangulation argument in "varsolve(,...) there are some hard to control approximations done consequent to the fact that the triangulations are not intersected. However the weights in the quadrature formula are divided by the number of quadrature points in Tk so as to obtain an exact formula for the integration of a constant.

3 Reserved words

3.1 Commands

```
1. " R2 "       " exit "

2. " if "      " then "     " else "
   " for "     " to "      " do "

3. " mesh "     " border "   " buildmesh "
   " savemesh " " readmesh " " movemesh "   " adaptmesh "

4. " function " " number"    " array"
   " plot "    " plot3d "  " save "
   " read "  " print " "append"

5." convect "

6. " dx "     " dy "       " laplace "
   " dxx "     " dxy "      " dyx "     " dyy "   " dnu "

7. " varsolve " " with "      " on "
   " solve "    " pde "

8  " int "     " Id "      " assemble " " derive "

9   " subroutine "
```

- "R2(x,y,ref)" means that the problem is 2D with variable x,y. Borders can be grouped into one unit by assigning to each the same "ref".
- "exit" is useful for program development.
- woX = convect(u,v,dt,w); means that w(x,y) is convected by the flow during a time interval dt: $woX(x,y) = w(X,Y)$:

$$\frac{dX}{dt} = u(X,Y), \quad \frac{dY}{dt} = v(X,Y), X(t-\delta t) = x; Y(t-\delta t) = y.$$

- All the operators like dx(u) can be used inside and outside PDEs; dnu(u) is not the normal derivative of u by its conormal.
- $Id(u > 1)$ is the Heavyside function; it is same as $u > 1$, only more readible.
- "assemble" and "derive" are for future uses.

3.2 Reserved variables

- "pi" = 4*atan(1.).
- "wait": if $wait \notin (-1, 0)$ graphics halt the program and the mouse must be clicked by the user. The variable *wait* controls also the way plots are displaid and stored (see the section on "plot").
- "nrmlx", "nrmly": components of the normal. (=0 if not on boundary)

4 adaptmesh

4.1 Syntax

The syntax of the language underlying freefem, which we call Gfem , is described by statements like

mesh $<$ **newMesh** $>=$ **adaptmesh**($<$ **oldMesh** $>, <$ **exp** $>$ [, $<$ **exp** $>]^{\mathbf{n}}$)
[$<$ **qualifier** $>$][, $<$ **qualifier** $>]^{\mathbf{n}}$

Here <exp> is any arithmetic expression involving functions or arrays and/or the coordinates (x,y in general) as explicit or implicit parameters.
The brackets [ ] means that the term is optional, the power n means that it can be repeated any number of time.

4.2 Example 1

```
border a(t=0,2*pi){x =cos(t); y = sin(t)};
mesh th = buildmesh(a(20));
mesh sh = adaptmesh (th,exp(x)/(0.01+y^2));
```

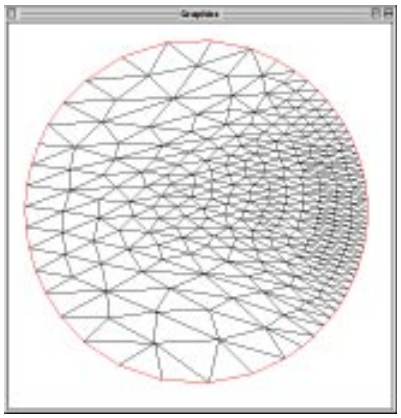

Figure 4
Mesh after adaption. The mesh before adaption can be seen on Figure 1. The following example adapt the mesh to the solution of a PDE which has a singularity due to an obtuse angle in the domain boundary.

4.3 Example 2

See file "adapt.edp"The domain Ω has the shape of an "L"

```
wait:=0 ;//tired of clicking mouse between graphs
border a(t=0,1){x=t;y=0};
border b(t=0,0.5){x=1;y=t};
border c(t=0,0.5){x=1-t;y=0.5};
border d(t=0.5,1){x=0.5;y=t};
border e(t=0.5,1){x=1-t;y=1};
border f(t=0,1){x=0;y=1-t};

mesh th = buildmesh(a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
```

The PDE is a simple Laplace equation with Dirichlet data

$$-\Delta u = 1, \quad \text{in} \quad \Omega, \quad u|_{\partial\Omega} = 0$$

To solve it we write

```
solve(u)  {
   pde(u)  laplace(u) = 1;
   on(a,b,c,d,e,f)  u=0;
  };
wait:=1; // back to the click mouse mode
plot(u);
```

Now we shall adapt the mesh several times with respect to u. The proper keywork is "adaptmesh". It takes for parameters the old mesh name and one or several function to which to the new mesh should be adapted.

```
err := 0.1;
coef := 0.1^(1./5.);

for i= 1 to 5  do
{
  err:=err * coef;
  print ("err=",err);
  mesh th = adaptmesh (th,u)
              verbosity=3,abserror=1,nbjacoby=2,
          err=err, nbvx=5000, omega=1.8,ratio=1.8, nbsmooth=3,
             splitpbedge=1, maxsubdiv=5,rescaling=1 ;

  solve(u)  {
     pde(u)  laplace(u) = 1;
     on(a,b,c,d,e,f)  u=0;
  };
  plot(u);
}
```

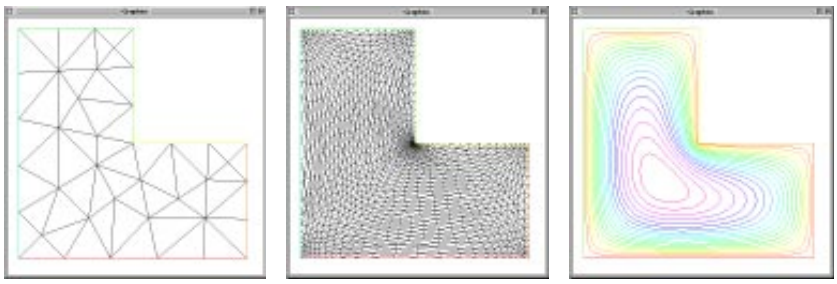

Figure 5
The initial mesh, the final mesh, and the solution of the PDE on the final mesh.

4.4 Example 3

See file "fitmesh.edp"This example illustrate the power of the adaption by hessian metrics, the method underlying the mesh generator and adaptor "bang" which is used in freefem+.
The idea is that the error of interpolation on a mesh is bounded by

$$\|u - u_h\| < C\|\nabla(\nabla u))\|h^2$$

Therefore an attempt to keep $\vec{h}^T\nabla(\nabla u)\vec{h}$ constant could pay. Bang does just that, and if you specify several function as input, like below it keeps $\vec{h}^T\nabla(\nabla s)\nabla(\nabla sy)\vec{h}$ constant. More precisely it applies the Delaunay-Voronoi triangulation algorithm with the distance function based on these Hessians (so that circles become ellpses). It is also substantially more complex because, as in the example below the third function,sz, has a null Hessian, but bang has made provisions for these degenerate cases.

```
wait := 0;
g(x,y) = 0.2*sin(x)*sin(x) + 0.02*y*y;

border a(t=0,pi*2){x = cos(t); y = sin(t)};
border b(t=0,2*pi){x = 0.3 + 0.3*cos(t);  y = 0.3*sin(t) };
mesh th = buildmesh(a(40) + b(-20)) ;

s=  (x^3+10*y^3) + 10/(1+10^(10*((sin(5*x)-2*y))));
sy =  (10*x^3+y^3) + 10/(1+10^(10*((sin(5*y)-2*x))));
sz := 100;

plot(s);
plot(sy());

for i= 1 to 2 do
{
mesh th = adaptmesh (th,sy,s,sz) verbosity=4,
```

```
                err=0.01, hmax=2, hmin=0.005,
                  nbvx=50000, omega=1.8, nbsmooth=0,
                  splitpbedge=0., maxsubdiv=5 ;
plot(s());
plot(sy());
};
```

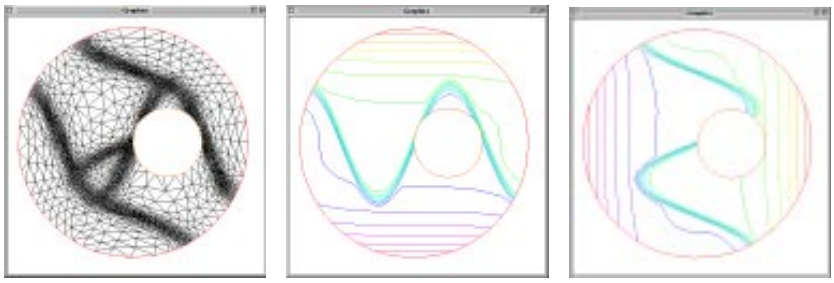

Figure 6
Final mesh after 2 iterations, and the level lines of s (center) and sx (left) that the mesh has adapted to.

5 Expressions

Freefem+ uses regular arithmetic expressions, like most programming languages. All expressions return a real value or an array of real values. As in C one can mixt booleans with reals, like $x * (y < 1)$. In addition there are new operators.

5.1 dx,dy

Basically dx(u) is the x-derivative of u. There is some provision for formal derivatives in the language but it is not yet released, so onyl arrays can be differentiated.

```
array u = x*y;
array v = dx(u);
plot(y-v); // should see zero
```

5.2 Int

Computes integrals. It can be part of an arthmetic expression.

```
a := int(mesh0)(f+u);
```

It can occur in any expression; the syntax is

$$\textbf{int}([< \textbf{mesh} >][\textbf{border}[,\textbf{border}]^{\textbf{n}}]) < \textbf{expression} >$$

- If no borders are specified it is a surface integral. Else it is a curve integral.
- Borders can be refered by their name or their reference number.
- If the mesh is not mentionned then the default mesh is used.
- The expression may involve arrays defined on another mesh.
- The quadrature uses the mid edge of the mesh/curve.
- The curve should be made of edges which are part of the mesh else the return value is zero, because freefem+ loops on the edges of the mesh and check if their label is in the list and add the contribution of that edge to the result.

Warning Just like any other expressions, integrals are computed when needed. Hence if you write

```
array a = x*y - int()(x*y);
```

the integral will be computed 250 times if there are 250 vertices!! You should write

```
b:=int()(x*y);
array a = x*y - b;
```

A similar bug, harder to catch:

```
f = x*y - int(x*y);
....
array a = f+1;
```

5.3 On

Used only with "varsolve"

5.4 Convect

This operator performs one step of convection by the method of Characteristics-Galerkin. An equation like

$$\partial_t \phi + u \nabla \phi = 0, \quad \phi(x,0) = \phi^0(x)$$

is approximated by

$$\frac{1}{\delta t}(\phi^{n+1}(x) - \phi^n(X^n(x))) = 0$$

Roughly the term,$\phi^n o X^n$ is approximated by $\phi^n(x - u^n(x)\delta t)$. Up to quadrature errors the scheme is unconditionnally stable.

5.4.1 Syntax

whose syntax is

$$< \textbf{array} >= \textbf{convect}(< \textbf{exp1} >, < \textbf{exp2} >, < \textbf{exp3} >, < \textbf{exp4} >)$$

- The "array" is the result uoX;
- "exp1" is the x-velocity,
- "exp2" is the y-velocity of convection,
- "exp3" is the time step,
- "exp4 is the expression which is convected (u in the exemple above)

Warning "convect" is a non local operator; in "phi=convec(u1,u2,dt,phi0)" every values of $phi0$ are used to compute phi So "phi=convec(u1,u2,dt,phi0)" won't work won't work.

5.4.2 Example 1

This a the rotating hill problem with one half turn. The surface $(x, y, z = \phi^0(x, y)$ looks like a hill. The velociy is a pure rotation so the hill rotates.

```
wait:=0;
border a(t=0, 2*pi){x := cos(t);y := sin(t);  };
mesh th = buildmesh(a(70)); // triangulates the unit disk
```

```
array phi0 = exp(-10*((x-0.3)^2 +(y-0.3)^2));
plot(v);

dt := 0.17; // time step
array u1 = y;
array u2 = -x; // rotation velocity

for i=0 to 20 do {
phi = convect(u1,u2,dt,phi0);
 phi0=phi;
plot(phi);
};
```

5.4.3 Example 2

In this example the same problem is solved but there is also some diffusion in addition to convection. The PDE is

$$\partial_t \phi + u\nabla\phi - \nu\Delta\phi = 0, \quad \phi(x,0) = \phi^0(x).$$

It is approximated by

$$\frac{1}{\delta t}(\phi^{n+1}(x) - \phi^n(X^n(x))) - \nu\Delta\phi^{n+1} = 0$$

```
nu := 0.01;
for i=0 to 20 do
{
solve(v) with A(i){
      pde(v) v/dt - laplace(v)*nu = convect(u1,u2,dt,v)/dt;
      on(a) v=0;
};
plot(v);
};
```

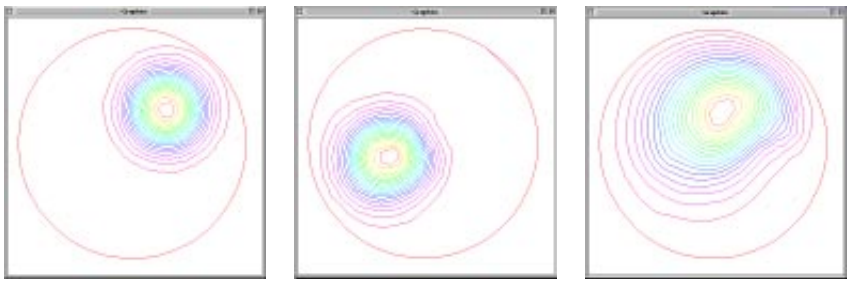

Figure 7
Initial condition, then pure convection (middle) after 20 steps and then 20 steps of convection-diffusion (right).

6 Solve

6.1 Syntax

$$\begin{aligned}
&\textbf{solve}([<\textbf{mesh}>,]<\textbf{array}>[,\textbf{array}]^{\textbf{n}})[\textbf{with}<\textbf{matrix}>(<\textbf{exp}>)]\\
\{&[<\textbf{instruction}>;]^{\textbf{n}}\textbf{pde}(<\textbf{array}>)[<\textbf{sign}>]<\textbf{operator}>[*|/<\textbf{exp}>]\\
&[<\textbf{sign}><\textbf{operator}>[*|/<\textbf{exp}>]]^{\textbf{n}}=<\textbf{exp}>;\\
&\textbf{on}(<\textbf{borderList}>)<\textbf{Dirichlet}>|<\textbf{Fourier}>=<\textbf{exp}>;\\
&[\textbf{on}(<\textbf{borderList}>)<\textbf{Dirichlet}>|<\textbf{Fourier}>=<\textbf{exp}>;]\\
&\};
\end{aligned}$$

In the above the | means "or" and whe have

- "mesh" is an existing mesh name
- "array" is either a new variable (which then becomes of array type or an existing array variable. There can be more than 1, up to 5 in the present implementation (it is easy to put more). These will be refered as "unknowns".
- "with" is useful if you intend to save computing time by reusing the matrix factorized. It is imperative to check that no left hand side has changed when you reuse a matrix, else the results will not be correct.
- "matrix" is a new variable or a variable previously declared as a matrix i.e. appearing at the same place in a solve statement earlier.

- the expression after "matrix" will be treated as a boolean (is it zero or not?)
- if necessary regular instructions can be inserted within the solve statement; it is usueful if they use the arrays declared after "solve".
- for each unknown there must be a pde and its boundary conditions.
- each pde is restricted expression made of operators applied to any of the unknowns; pde(u) can involve dx(v)....
- "borderList" is a list of names of borders separated by a comma and/or a list of integers which are the references to a set of borders.

$$< \textbf{Dirichlet} > \equiv \quad < \textbf{array} > = < \textbf{exp} >$$
$$< \textbf{Neumann} > \equiv \quad [< \textbf{sign} >] < \textbf{op} > [*|/ < \textbf{exp} >]$$
$$[< \textbf{sign} >< \textbf{op} > [*|/ < \textbf{exp} >]]^{\textbf{n}} = < \textbf{exp} >$$

where "op" is a boundary operator, i.e. either any unknown or dnu(¡theUnknown¿) and "theUnknown" is xxx the variable inside pde(xxx).

6.2 Example 1: the Stokes problem

The driven cavity flow problem is solved first at zero Reynolds number The velocity pressure formulation is used first and then the calculation is repeated with the stream function vorticity formulation. The domain is the unit square

```
border a(t=0,1){x=t; y=0};
border b(t=0,1){x=1; y=t};
border c(t=1,0){x=t; y=1};
border d(t=1,0){x=0; y=t};
mesh th= buildmesh(a(20)+b(20)+c(20)+d(20));
```

Then the system for the velocity $\vec{u} = (u, v)$ and the pressure p is

$$-\Delta\vec{u} + \nabla p = 0, \quad \nabla \cdot \vec{u} = 0$$

and the boundary conditions are zero velocities except on the top boundary where $u = 1$. regularization is necessary to avoid checkerboard oscillations so the div condition is replace by

$$-\epsilon\Delta p + \nabla \cdot \vec{u} = 0$$

We do not intend to reuse the matrix so,

```
solve(u,v,p){
     pde(u)  - laplace(u) + dx(p) = 0;
     on(a,b,d) u =0;
     on(c) u = 1;
     pde(v) - laplace(v) + dy(p) = 0 ;
     on(a,b,c,d) v=0;
     pde(p) p*0.001- laplace(p)*0.001 + dx(u)+dy(v) = 0;
     on(a,b,c,d) dnu(p)=0;
};
```

The streamlines are the level line of the function ψ such that $\nabla \times \psi = u$, an equation which when differentiated lead to

$$-\Delta(\psi) = \partial_y u - \partial_x v.$$

```
// show stream lines
solve(psi){ pde(psi) -laplace(psi) = dy(u)-dx(v);
         on(a,b,c,d) psi=0};
plot(psi);
```

Now let us solve the same problem in psi-omega formulation

$$\Delta\psi = \omega, \quad \Delta\omega = 0; \quad \psi|_\Gamma = 0, \quad \partial_\nu\omega|_\Gamma = g$$

```
solve(psi,om){
pde(psi)  om -laplace(psi) = 0;
on(a,b,d) dnu(psi)=0;
on(c) dnu(psi) = 1;
pde(om)  - laplace(om) = 0;
on(a,b,c,d) dnu(om) + psi*1e8 = 0;
};
```

Notice the last boundary condition which is a trick to impose $\psi = 0$. The problem is that we can only write boundary conditions on ω at this level,everything we could write on the level of the ψ equation has been done (one condition only per boundary). So by adding a tiny term $\partial_\nu \omega$ to the equation $\psi = 0$, the trick is plaid.

6.3 Example 2: Navier-Stokes equations

The Navier-Stokes equations offer an opportunity to illustrate the reusability of matrices.

$$\partial_t \vec{u} + \vec{u} \cdot \nabla \vec{u} - \nu \Delta \vec{u} + \nabla p = 0, \quad \nabla \cdot \vec{u} = 0$$

is approximated in time by

$$\frac{1}{\delta t}(u^{n+1} - u^n o X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \quad \nabla \cdot u^{n+1} = 0$$

The term,$u^n o X^n(x) \approx u^n(x - u^n(x)\delta t)$ will be computed by the operator "convect", so we obtain

```
nu:=0.01; dt :=0.1;
for i=0 to 20 do
{
solve(u,v,p) with B(i){
pde(u)  u/dt- laplace(u)*nu + dx(p) = convect(u,v,dt,u)/dt;
on(a,b,d) u =0;
on(c) u = 1;
pde(v) v/dt- laplace(v)*nu + dy(p) =  convect(u,v,dt,v)/dt;
on(a,b,c,d) v=0;
pde(p) p*0.1*dt - laplace(p)*0.1*dt + dx(u)+dy(v) = 0;
on(a,b,c,d) dnu(p)=0;
};
plot(u);
};
```

Notice that the first time solve occurs it has B(0) so the matrix is built and factorize. The second time it has B(1) so the matrix is reused.

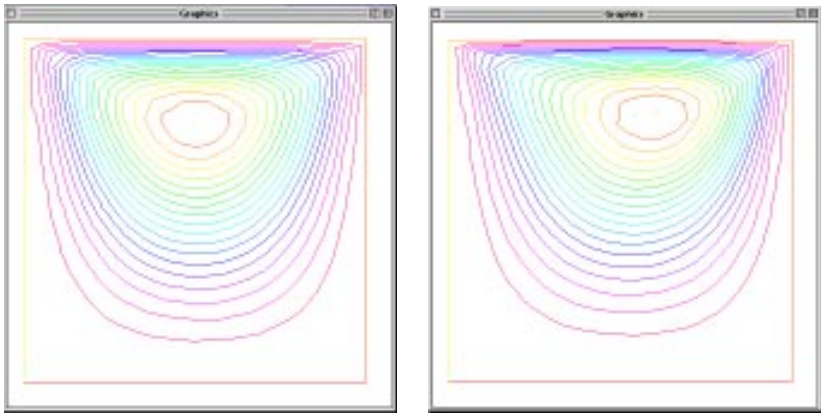

Figure 8
Stream lines of the Stokes flow (left) and Navier-Stokes flow after 20 time steps.

6.4 Example 3: The Backward Step Problem

The flow in an expanding pipe is studied. It is again the Navier-Stokes The Reynolds number based on the size of the step an the mean inflow is Re = 200. The projection algorithm is used as in Rannacher-Turek (featflow).

```
wait := 0;
n:=3;
border a(t=0,1)  {x=0;      y=1-t   };
border b(tp = 0,4) { if(tp<1)then  { t=tp;  x=2*t;    y=0     }

  else if((tp>=1)*(tp<2)) then { t = tp-1; x=2;      y=-t    }
  else if((tp>=2)*(tp<3)) then { t = tp-2; x=2+6*t;  y=-1    }
else { t = tp-3; x=8+12*t; y=-1    }
};
border c(t=0,1) {x=20;     y=-1+2*t };
border d(tp=0,2) { if(tp<1)then { t=tp;    x=8+12*(1-t);  y=1 }
else { t = tp-1;  x=8*(1-t); y=1     }
 };
mesh th = buildmesh( a(3*n) + b(50*n) + c(5*n) + d(36*n) );
```

```
nu := 0.005; dt := 0.1;
area:= int()(1.);
array ub = 4*y*(1-y)*(y>0);

array u = 0;
array v = 0;
array p = 0;

for i=0 to 50 do
{

solve(u) with A(i){
pde(u) u/dt - laplace(u)*nu =convect(u,v,dt,u)/dt - dx(p);
on(a) u =ub;
on(b,d) u=0;
on(c) u=convect(u,v,dt,u);
};
plot(u);

solve(v) with B(i){
pde(v) v/dt - laplace(v)*nu = convect(u,v,dt,v)/dt - dy(p);
on(a,b,d) v=0;
on(c) v=0;
};

qq := int()(dx(u)+dy(v))/area;
solve(q) with C(i){
pde(q) q*0.01*dt- laplace(q)*dt = dx(u)+dy(v)-qq;
on(c)q=0;
};

p = p - q;
pp := int()(p)/area;
p = p-pp;
u = u + dx(q)*dt;
v = v + dy(q)*dt;
 } ;
```

```
wait:=1;
plot(u);
```

Figure 9: The horizontal velocity after 100 time step.

7 Varsolve

This keyword triggers a very powerful feature of freefem+ where by the linear system and right hand side of a variational equation is constructed automatically and solved.

7.1 Syntax

$$\textbf{varsolve}[(<\textbf{mesh}>[,<\textbf{exp}>])]<\textbf{ident}>(<\textbf{ident}>,<\textbf{ident}>[,<\textbf{ident}>,<\textbf{ident}>]^{\textbf{n}})\textbf{with}<\textbf{instruction}>;$$

Example:
The following variational form

$$\int_\Omega(\partial_x u\partial_x w+\partial_y u\partial_y w-w)+\int_{\Gamma_a}uw=\int_\Omega w+\int_{\Gamma_a}gw,$$

$$\forall w\ :w|_{\Gamma_b\cup\Gamma_c}=0\quad\text{with } u|_{\Gamma_b\cup\Gamma_c}=u_\Gamma$$

is coded as

```
varsolve(th,i) aa(u,w) with {
aa = int()( dx(u)*dx(w)+dy(u)*dy(w)-w)
 + int(a)(w*(u-g))
+ on(b,c)(w)(u=ugamma);
};
```

If $i=0$ it is used for the first time, else it is the second time it is used and the right hand side of the variational formula (the theoretical one) has changed only.
The following should be noted:

- If no mesh is specified, the default mesh is used. If the mesh is specified it become the default mesh within the "instruction".
- The list of "iden" betwen parenthesis is the variable list and the hat function list. Each variable is followed by its associated hat function.
- The "instruction" should define the "ident". It is expected to be a bilinear form in the variables. If it is not some kind of unpredictable linearization with go on.
- The "instruction" can be a block of instructions.
- To construct the matrix and linear system what freefem+ does is to loop on each triangle, treat all variables as hat functions, assign to the hat functions a "1" on one vertex, "0" elsewhere and compute the value of the aa.
- The operator "on" is a penalty operator, in this exemple it returns $10^7 w(q^j)(u(q^j) - u_\Gamma(q^j))$ at vertex q^j.

7.2 Example 1: Domain decomposition

Suppose an object Ω is made on two parts $\Omega = \Omega_1 \cup \Omega_2$, $\Omega_1 \cap \Omega_2 \neq 0$ and we have a mesh for both parts but no mesh for the whole. We can still do by the Schwarz algorithm. The idea is to compute the solution on one domain and use the value of this computation for the missing boundary condition for the computation on the other domain.

```
// Solution by Schwarz algorithm
wait:=0;
border a(t=0,1){x=t;y=0};
border a1(t=1,2){x=t;y=0};
border b(t=0,1){x=2;y=t};
border c(t=2,0){x=t ;y=1};
border d(t=1,0){x = 0; y = t};
border e(t=0, pi/2){x= cos(t); y = sin(t)};
border e1(t=pi/2, 2*pi){x= cos(t); y = sin(t)};
n:=4;
//Omega1
mesh th = buildmesh( a(5*n) + a1(5*n) + b(5*n) + c(10*n) + d(5*n));
//Omega2
```

```
mesh TH = buildmesh ( e(5*n) + e1(25*n) );
//Omega1+Omega2 (only to compute the error)
mesh sh = buildmesh (a1(5*n) + b(8*n) + c(10*n) + e1(25*n));

// usual FEM solution
varsolve(sh,0) aa(uu,ww) with {
aa = int()( dx(uu)*dx(ww)+dy(uu)*dy(ww) - ww )
+ on(a1,b,c,d,e1)(ww)(uu=0);
};
plot(sh,uu);

array(TH) uold=0;
array(th) Uold=0;

CHI = (x^2+y^2) <= 1.0;
chi = (x>=-0.01)*(y>=-0.0)*(x<=2.0)*(y<=1.0);

for i=0 to 10 do
{
varsolve(TH,i) AA(U,W) with {
AA = int(TH)( dx(U)*dx(W)+dy(U)*dy(W)-W)
+ on(e,e1)(W)(U=uold*chi);
};

varsolve(th,i) aa(u,w) with {
aa = int(th)( dx(u)*dx(w)+dy(u)*dy(w)-w)
+ on(a,a1,b,c,d)(w)(u=Uold*CHI);
};
Uold = U*CHI;
uold = u*chi;
print("error=",int(th)((u-uu)^2 + (dx(u)-dx(uu))^2+(dy(u)-dy(uu))^2)
 + int(TH)((U-uu)^2 + (dx(U)-dx(uu))^2+(dy(U)-dy(uu))^2) );
};

// display error
plot(sh,uold+Uold-(uold+Uold)*chi*CHI/2-uu);
```

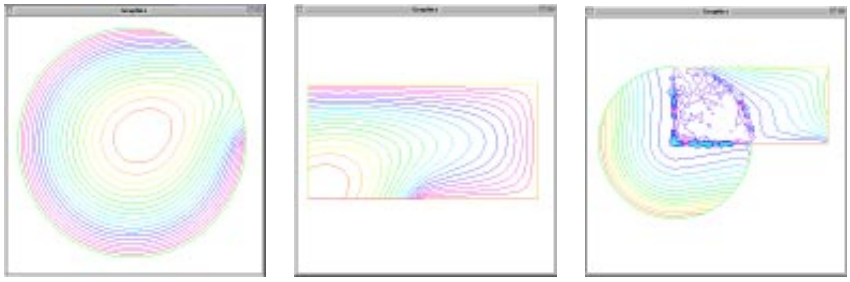

Figure 10
The solution has been computed separately on the circle and one the square. At their intersection, they match, the plot shows that the error is zero there.

7.2.1 Extension

One important application of the Schwarz algorithm is the "Chimera" method in CFD [**?**]. Then the domain of computation being the outside of given closed bounded sets (objects), to apply Schwarz algorithm one needs to surround each object by a computational subdomain which overlaps the mesh of other domains. This can be done automatically with the introduction of piecewise constant functions (P0) and a domain identification procedure via the characteristic functions of P0 functions. This will be described in greater details in a future publication but an example of application is given below:

```
wait:= 1;  verbosity=1;
border a(t=0,1){x=t;y=0};
border b(t=0,0.5){x=1;y=t};
border c(t=0,0.5){x=1-t;y=0.5};
border d(t=0.5,1){x=0.5;y=t};
border e(t=0.5,1){x=1-t;y=1};
border f(t=0,1){x=0;y=1-t};
i:=0;  n:=4;
mesh Th0 = buildmesh(a(24)+b(16)+c(16)+d(16)+e(16)+f(24));
border circle(t=0,2*pi){x=cos(t)/4;y=sin(t)/4};
mesh Th1 = buildmesh ( circle(200));

femp0(Th1)  Chi1 = 1; // Characteristic function of mesh 1
femp1(Th0)  k1 = Chi1;
femp1(Th0)  K = k1;
```

```
plot(Th0,k1);

femp0(Th0)  k0=k1;
plotp0(Th0,k0>0);
k1=k0;   plot(Th0,k1);
plotp0(Th0,k1-k0);
k0=k1;  plotp0 (Th1,1,Th0,k0>0);
mesh Thr = Th0(k0>0,2);
plot (Thr,1,Th1,2); // two plots on one screen
```

7.3 Example2: Fluid-Structure Interactions

We come back to the driven cavity with the Stokes equations of Example 1 in the section on the keyword "solve". We add to it a plastic lead which deforms under its own weight. The box of fluid "sh" with its lead th" "are triangulated as

```
border a(t=2,0)  {x=0; y=t };
border c(t=0,2)  {x=10; y=t };
border d(t=0,10) {x=10-t; y=2 };
border e(t=0,10) {x=t; y=-10 };
border f(t=0,10) {x=10; y=-10+t };
border g(t=0,10) {x=0; y=-t };
border b(t=0,10) {x=t; y=0 };
mesh sh = buildmesh(b(-20)+f(15)+e(15)+g(15));
mesh th = buildmesh( b(40)+c(20)+d(40)+a(20));
```

The equations of elasticity are naturally written in variational form for the displacement vector $u(x) \in V$ as

$$\int_\Omega [\mu \epsilon_{ij}(\vec{u})\epsilon_{ij}(\vec{w}) + \lambda tr(\epsilon(u))tr(\epsilon(w)] = \int_\Omega f \cdot w + \int_\Gamma h \cdot w, \forall w \in V$$

where "tr" is the trace operator and with

$$\epsilon_{ij}(u) = \frac{1}{2}(\partial_i u_j + \partial_j u_i)$$

The space V may contain constraints of Dirichlet type if the structure is clamped on part of its boundary. Here we assume that the lead is clamped on its vertical sides and that the only force is the gravity. Then the system is solved by

```
varsolve(th,0) bb(uu,w,vv,s) with {
e11 = dx(uu);
e22 = dy(vv);
e12 = (dx(vv)+dy(uu))/2;
w11 = dx(w);
w22 = dy(s);
w12 = (dx(s)+dy(w))/2;
bb = int()( 2*mu*(e11*w11+e12*w12+e22*w22)
            + lambda*(e11+e22)*(w11+w22)/2 -gravity*s)
 + on(a,c)(w)(uu=0)
 + on(a,c)(s)(vv=0)
};
mesh th1 = movemesh(th, x - uu, y - vv);
```

The last line allows us to see the deformed structure. As we intend to reuse the matrix of the linear system and avoid factorization, we use a zero in the parameter.
When the fluid rotates the lid of the cavity is subject to the normal stress due to the fluid.

$$\sigma n = \nabla u + \nabla u^T - pn$$

This surfacic force acts on the structure and deforms it. Now to include the effect of the fluid surface stress we solve

```
varsolve(1) bb(uu,w,vv,s) with {
bb = int()( 2*mu*(e11*w11+e12*w12+e22*w22)
             + lambda*(e11+e22)*(w11+w22)/2 -gravity*s)
           + coef*int(b)( (2*dx(u)-p)*nrmlx*w + (2*dy(v)-p)*nrmly*s
             + (dx(v)+dy(u))*(nrmly*w + nrmlx*s))
 + on(a,c)(w)(uu=0)
 + on(a,c)(s)(vv=0)
};
```

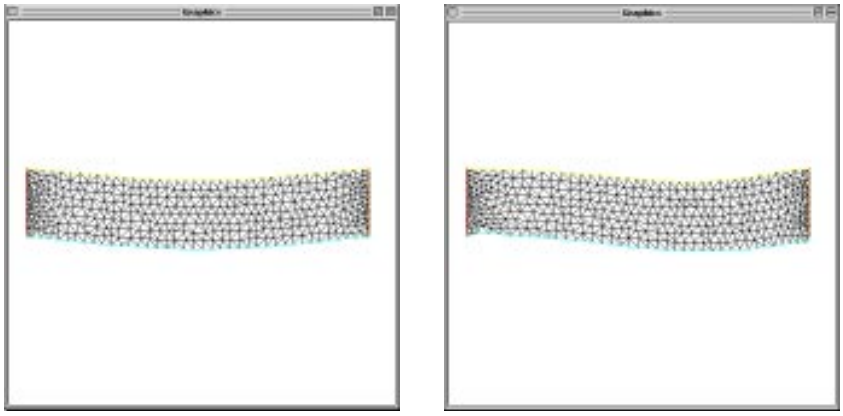

Figure 11
Deflection of a beam by its own weight (left) and by its weight and the viscous and pressure effect of a Stokes flow.

8 Graphics and files

8.1 The keyword "plot"

This instruction will display the level lines of one or more functions. The syntax is:

plot([< **filename** >,][< **mesh** >,] < **expression** > [, [< **mesh** >,] < **expression** >]$^{\mathbf{n}}$)

The "mesh" is the one which determines the domain on which the function is plotted. It does not have to be the one on which the function(s) is(are) defined.
By translations (movemesh) one can control the layout of the plots.

8.1.1 Zooms and plots

Graphics may flow continuously or may halt the program to allow the user to study them. If $wait \in [-1, 0]$ graphics do not halt the program. The "graphic window" is initialized at startup so that the domain fits and is centered in the graphic window. If wait is active (i.e. $wait \notin [-1, 0]$) then while a graphic is displaid the user can redefine the layout of the graphic in the graphic window by typing $+, -$ or $=$.

If + (resp -) is typed the graphic is zoomed (resp unzoomed) around the position of the cursor. If = is typed then the graphic position is reinitialzed to what it was at startup.

8.1.2 Plots and the variable "wait"

If *wait* is active then it is possible to redefine it by typing

- r 0 : then no subsequent graphics will halt the program
- r 1 : all subsequent graphics will halt the program
- r -1 no subsequent graphics will halt the program and the graphic window is reinitialized to its values at startup.
- r -2 : all subsequent graphics will halt the program and the graphic window is reinitialized to its values at startup.

8.2 Summary of keyboard commands

If *wait* is active only:

- <Control> c : stops the program
- r : redraws the graphic
- r and 1,0,-1,-2: redraws the graphic and redefines "wait" to the typed value (if in wait active mode)
- + : zoom in the current graphic and all subsequent ones (if in wait active mode)
- - : unzoom the current graphic and all subsequent ones (if in wait active mode)
- = : reinitialize the current graphic and all subsequent ones (if in wait active mode)

8.3 The keywords "readmesh", "savemesh","read" and "save"

There are a number of formats to save meshes and arrays and freefem respond to the suffixe of the file name given.

$$\textbf{savemesh}(\text{" } < \textbf{filename} > \text{"}[, < \textbf{meshname} >])$$

The string is the file name and if no mesh is specified then the default mesh is used. The filename is xxx or xxx.xxx where xxx is any number of alphanumeric character. The last part after the "." is the suffix, if any. By default the format is the same as in freefem 3.0.

$$\textbf{mesh} < \textbf{name} >= \textbf{readmesh}(\text{" } < \textbf{filename} > \text{"})$$

Mesh file formats are

- .amdba the INRIA amdba format
- .msh the freefem format
- .am_fmt the EMC2 INRIA format

Note triangulations are not renumbered by the read statement. Labels for boundaries remain numerical but they can be used as label in statements like "on(4,5)....".

Other data file format are

- .dbg an extended talkative format for debugging (savemesh only)
- .gnu a format to plot 3D surfaces with gnuplot "with lines". (save only)
- .ps a format to save the equivalent graphic (with its zoomed options) in PostScript

$$\textbf{save}(\text{" } < \textbf{filename} > \text{"}, < \textbf{arrayname} >)$$

Saves in ¡filename¿ a function named ¡arrayname¿ on the active triangulation.

$$\textbf{read}(\text{" } < \textbf{filename} > \text{"}, < \textbf{arrayname} >)$$

Reads from ¡filename¿ a function named ¡arrayname¿ on the active triangulation.

8.4 Save displaid plots

This section applies to the 3 keywords "plot", "buildmesh", "adaptmesh".

It is useful to be able to save graphics as displaid, in PostScript format. This can be done by putting a file name in the parameter list of the instruction plot.
Example
plot("f.ps",th,f);

It is also useful to save a graphic withing an iteration loops. Then the filename must change at each loop. There is provision for this by using a file name in the format

$$" < \textbf{string} > " < \textbf{var} > " < \textbf{string}"$$

Example

```
for i=0 to 10 do
{
  varsolve(TH,i) AA(U,W) with {....} // defines U

  plot(U);
  save("U_"i".ps",U);
}
```

9 Application: Domain Decomposition

We recently ran into the following problem: is it possible to solve a PDE in a domain $\Omega - O$ where O is a hole in the domain without ever having to generate the triangulation of $\Omega - O$?
Here is an answer (Lions et al [1]).
Let $V = H_0^1(\Omega)$. we look for $u \in V$ such that

$$\int_\Omega \nabla u \nabla v = \int_\Omega fv \quad \forall v \in V$$

Assume that Ω ait un trou O. Let $\Omega_1 = \Omega$ and let Ω_2 be an open set that contains O and is contained in Ω_1 Let $V_i = H_0^1(\Omega_i)$.

Algorithm Loop on $u_1^{n+1} \in V_1$ such that

$$\int_{\Omega_1} \nabla u_1^{n+1} \nabla v_1 + \int_{\Omega_{12}} \nabla u_2^n \nabla v_1 = \int_{\Omega_1} f v_1, \quad \forall v_1 \in V_1$$

and loop on $u_2^{n+1} + u_1^n \in V_2$ such that

$$\int_{\Omega_2} \nabla u_2^{n+1} \nabla v_2 + \int_{\Omega_{12}} \nabla u_2^n \nabla v_2 = \int_{\Omega_2} f v_2, \quad \forall v_i \in V_i$$

The following program is an application of this idea to a schematic contert hall (Helmholtz equation) where rows of seats are inserted. The mesh of the concert hall with the seats is not needed except to check the results.

```
wait:=0;
border a(t=0,1){x=5; y=1+2*t};
border b(t=0,1){x=5-2*t; y=3};
border c(t=0,1){x=3-2*t; y=3-2*t};
border d(t=0,1){x=1-t; y=1};
border f(t=0,1){x=0; y=1-t};
border g(t=0,1){x=t; y=0};
border h(t=0,1){x=1+4*t; y=t}; //lower part inclined bdy
border h1(t=0,1){x=1; y=0.4*t};
border h2(t=0,1){x=1+4*t; y=0.4+t};
border a1(t=0,0.8){x=5; y=1.4+2*t};

border e1(t=pi*2,0){x=1.5+0.08*cos(t); y= 0.1+0.3+0.14*sin(t)};
border e2(t=pi*2,0){x=2.5+0.08*cos(t); y= 0.1+0.6+0.14*sin(t)};
border e3(t=pi*2,0){x=3.5+0.08*cos(t); y= 0.1+0.85+.14*sin(t)};
border e4(t=pi*2,0){x=4.5+0.08*cos(t); y= 0.1+1.1+0.14*sin(t)};
border f1(t=0,pi*2){x=1.5+0.16*cos(t); y= 0.1+0.35+.28*sin(t)};
border f2(t=0,pi*2){x=2.5+0.16*cos(t); y= 0.1+0.6+0.28*sin(t)};
border f3(t=0,pi*2){x=3.5+0.16*cos(t); y= 0.1+0.85+.28*sin(t)};
border f4(t=0,pi*2){x=4.5+0.16*cos(t); y= 0.1+1.1+0.28*sin(t)};

n:=5;
mesh sh = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(7*n)+ g(5*n) + h(10*n)
+e1(15)+e2(15)+e3(15)+e4(15));
function ff(x,y)  =   0.045 >( (x-0.45)^2 + (y-0.45)^2);
plot(sh,2*ff(x,y));
```

```
mesh TH = buildmesh ( a(5*n) + b(5*n)
         + c(10*n)+d(7*n)+f(5*n)+g(5*n) + h(10*n) );

mesh th1 = buildmesh (e1(50)+f1(50));
mesh th2 = buildmesh (e2(50)+f2(50));
mesh th3 = buildmesh (e3(50)+f3(50));
mesh th4 = buildmesh (e4(50)+f4(50));

mesh sh;
om := 6;
varsolve(sh,0) aa(uu,ww) with {
aa = int()(om*uu*ww- dx(uu)*dx(ww)-dy(uu)*dy(ww) - ff(x,y)*ww )
+ on(a,b,c,d,f,g,h,e1,e2,e3,e4)(ww)(uu=0);
};
plot(sh,uu);

array(TH) uold4=0;
array(TH) uold3=0;
array(TH) uold2=0;
array(TH) uold1=0;
array(TH) Uold=0;

for i=0 to 20 do
{
varsolve(TH,i) AA(U,W) with {
AA = int(TH)(om*(U+uold1+uold2+uold3+uold4)*W
  - dx(U)*dx(W)-dy(U)*dy(W)-ff(x,y)*W
-dx(uold1)*dx(W)+dy(uold1)*dy(W)
-dx(uold2)*dx(W)+dy(uold2)*dy(W)
-dx(uold3)*dx(W)+dy(uold3)*dy(W)
-dx(uold4)*dx(W)+dy(uold4)*dy(W) )
+ on(a,b,c,d,f,g,h)(W)(U=0);
};
Uold  = (U +Uold)/2;

varsolve(th1,i) aa1(u1,w1) with {
aa1 = int()( om*u1*w1-dx(u1)*dx(w1)-dy(u1)*dy(w1)-ff(x,y)*w1
+om*Uold*w1 - dx(Uold)*dx(w1)-dy(Uold)*dy(w1))
```

```
+ on(e1)(w1)(u1=-Uold)+ on(f1)(w1)(u1=0);
};
varsolve(th2,i) aa2(u2,w2) with {
aa2 = int()( om*u2*w2-dx(u2)*dx(w2)-dy(u2)*dy(w2)-ff(x,y)*w2
+om*Uold*w2 - dx(Uold)*dx(w2)-dy(Uold)*dy(w2))
+ on(e2)(w2)(u2=-Uold)+ on(f2)(w2)(u2=0);
};
varsolve(th3,i) aa3(u3,w3) with {
aa3 = int()( om*u3*w3-dx(u3)*dx(w3)-dy(u3)*dy(w3)-ff(x,y)*w3
+om*Uold*w3 - dx(Uold)*dx(w3)-dy(Uold)*dy(w3))
+ on(e3)(w3)(u3=-Uold)+ on(f3)(w3)(u3=0);
};
varsolve(th4,i) aa4(u4,w4) with {
aa4 = int()( om*u4*w4-dx(u4)*dx(w4)-dy(u4)*dy(w4)-ff(x,y)*w4
+om*Uold*w4 - dx(Uold)*dx(w4)-dy(Uold)*dy(w4))
+ on(e4)(w4)(u4=-Uold)+ on(f4)(w4)(u4=0);
};
uold1 = (u1+uold1)/2;
uold2 = (u2+uold2)/2;
uold3 = (u3+uold3)/2;
uold4 = (u4+uold4)/2;
plot(sh,uold1+uold2+uold3+uold4);
append("error=",
 int(sh)((uold1+ uold2+uold3+uold4+Uold-uu)^2
+(dx(uold1)+ dx(uold2)+dx(uold3)+dx(uold4)+dx(Uold)-dx(uu))^2
+(dy(uold1)+ dy(uold2)+dy(uold3)+dy(uold4)+dy(Uold)-dy(uu))^2
));
plot(sh,uold1+uold2+uold3+uold4+Uold-uu);
};
```

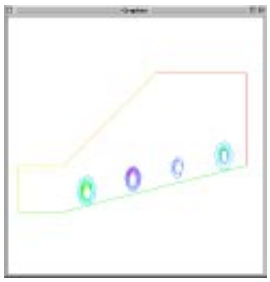
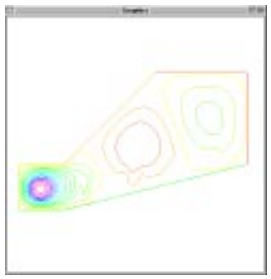
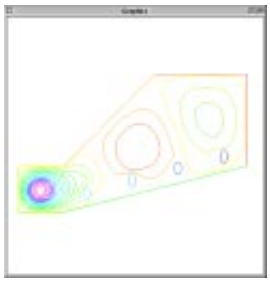

Figure 12

The full solution (right) is obtained by superposition of the local solutions near the holes (left) withthe solution in the big domain without the holes (middle).

10 References

1. J.L. Lions, O. Pironneau: Superpositions for composite domains (to appear)
2. B. Lucquin, O. Pironneau: *Scientific Computing for Engineers* Wiley 1998."
3. P.L. George: *Automatic triangulation*, Wiley 1996"
4. "F. Hecht: The mesh adapting software; bang. INRIA report 1998."
5. Rannacher-Turek (featflow)

11 Appendix A: Example of line output (Schwarz algorithm)

```
Welcome to freefem+
        Start Program FreeFem+0.9.31
wait:=0;
 border a(t=0,1){x=t;
y=0};
 border a1(t=1,2){x=t;
y=0};
 border b(t=0,1){x=2;
y=t};
 border c(t=2,0){x=t;
y=1};
 border d(t=1,0){x=0;
y=t};
 border e(t=0,pi/2){x=cos(t);
y=sin(t)};
 border e1(t=pi/2,2*pi){x=cos(t);
y=sin(t)};
```

```
n:=4;
 mesh th=buildmesh ("th1",a(5*n)+a1(5*n)+b(5*n)+c(10*n)+d(5*n));
         Save Postscript in file 'th1.ps'
 th2t 1 1
        set the active mesh th g = 0x02692f1c
 mesh TH=buildmesh ("th2",e(5*n)+e1(25*n));
         Save Postscript in file 'th2.ps'
 th2t 1 1
        set the active mesh TH g = 0x026898ac
 mesh sh=buildmesh (a1(5*n)+b(8*n)+c(10*n)+e1(25*n));
 th2t 1 1
        set the active mesh sh g = 0x0272bbd8
 varsolve (sh,0)aa(uu,ww) with {aa= int ()( dx (uu)* dx (ww)
        + dy (uu)* dy (ww)-ww)+ on (a1,b,c,d,e1)(ww)(uu=0);
};
         pivot= 2.2805
 plot (sh,uu);
 array (TH)uold=0;
 array (th)Uold=0;
CHI=(x^2+y^2)<=1;
chi=(x>=-0.01)*(y>=-0)*(x<=2)*(y<=1);
 for i=0 to 5 do { varsolve (TH,i)AA(U,W) with
       {AA= int (TH)( dx (U)* dx (W)+ dy (U)* dy (W)-W)
            + on (e,e1)(W)(U=uold*chi);
};
 varsolve (th,i)aa(u,w) with {aa= int (th)( dx (u)* dx (w)
       + dy (u)* dy (w)-w)+ on (a,a1,b,c,d)(w)(u=Uold*CHI);
};
Uold=U*CHI;
uold=u*chi;
 print ("error=", int (th)((u-uu)^2+( dx (u)- dx (uu))^2
     +( dy (u)- dy (uu))^2)+ int (TH)((U-uu)^2+( dx (U)- dx (uu))^2
     +( dy (U)- dy (uu))^2));
};
##########  iteration 0 ------------
         pivot= 2.27424
         pivot= 2.32272
error= 0.155605
##########  iteration 1 ------------
```

```
error= 0.0139767
##########  iteration 2 ------------
error= 0.00473086
##########  iteration 3 ------------
error= 0.00379549
##########  iteration 4 ------------
error= 0.00373622
##########  iteration 5 ------------
error= 0.00373999
wait:=1;
 plot ("u2",TH,U);
         Save Postscript in file 'u2.ps'
 plot ("u1",th,u);
         Save Postscript in file 'u1.ps'
 plot ("err",sh,uold+Uold-(uold+Uold)*chi*CHI/2-uu);
         Save Postscript in file 'err.ps'
 end No Error
Normal exit 0
```