Solving PDEs with finite elements: an introduction to free software FreeFem++ Part I: Basic features

Ionut Danaila

University of Rouen Normandie, France

ionut.danaila.perso.math.cnrs.fr

Institute of Mathematical Sciences, Singapore.
November 13, 2019.

Basic features in FreeFem++

Archive to be downloaded:
http://ionut.danaila.perso.math.cnrs.fr/zdownload/FFEM/

1. **WHAT is FreeFem++?**
2. **WHY using FreeFem++?**
3. **HOW to use FreeFem++ (a step-by-step guide)**
   - (1) How to install FreeFem++?
   - (2) Building a mesh
   - (3) What mathematics do you need to know?
   - (4) Solving the Poisson equation in 10 lines of code
   - (5) Dealing with Boundary Conditions
4. **Summary of basic features.**

What is FreeFem++?

Intuitive answer

...yet another finite-element software!

What is FreeFem++?

Intuitive answer

...yet another finite-element software!

New answer (after this course)

...THE finite-element software you need!

What is FreeFem++?

Intuitive answer

...yet another finite-element software!

New answer (after this course)

...THE finite-element software you need!

FreeFem++ (www.freefem.org)

Free Generic PDE solver using finite elements (2D and 3D)

- syntax close to the mathematical (weak) formulations,
- powerful mesh generator,
- mesh interpolation and adaptivity,
- use combined P1 to P4 Lagrange elements, Raviart-Thomas, etc,
- complex matrices,
- parallel computing (domain decomposition methods), etc.

Why using FreeFem++?

Free

- research
- industry

Easy to use

steep learning curve

Modern

interface to up-to-date libraries

Close to maths

low effort to implement complex methods

Why using FreeFem++?

Free

- research
- industry

Easy to use

steep learning curve

Modern

interface to up-to-date libraries

Close to maths

low effort to implement complex methods

Why using FreeFem++?

Free

- research
- industry

Easy to use

steep learning curve

Modern

interface to up-to-date libraries

Close to maths

low effort to implement complex methods

Why using FreeFem++?

Free

- research
- industry

Easy to use

steep learning curve

Modern

interface to up-to-date libraries

Close to maths

low effort to implement complex methods

You know what you do and keep control on

- algorithms/methods,
- parameters, convergence criteria, etc.

Why using FreeFem++?

Free
- research
- industry

Easy to use

steep learning curve

Modern

interface to up-to-date libraries

Close to maths

low effort to implement complex methods

You know what you do and keep control on
- algorithms/methods,
- parameters, convergence criteria, etc.

Large community (Europe, Japan, China, Canada, etc)

You are welcome to participate in the:

FreeFem++ Days, Paris, December, every year.

Utilisation of FreeFem++

Physics	Numerical meth.	Implementation	Results
obs/equations	PDE/num analysis.	algorithm/code	physical detail

Utilisation of FreeFem++

Physics	Numerical meth.	Implementation	Results
obs/equations	PDE/num analysis.	algorithm/code	physical detail

Solve complicated PDEs/ Post-processing of results

- avoid technicalities of the FE-method,
- obtain rapidly numerical results,
- initiate collaborations with physics and industry.

Utilisation of FreeFem++

Physics	Numerical meth.	Implementation	Results
obs/equations	PDE/num analysis.	algorithm/code	physical detail

Solve complicated PDEs/ Post-processing of results

- avoid technicalities of the FE-method,
- obtain rapidly numerical results,
- initiate collaborations with physics and industry.

Develop/Test new numerical methods

- write lines of code like writing mathematical equations,
- versatile (easy-to-change) scripting (change the type of FE, preconditioner, linear solver, etc),
- check mathematical theories (theory of PDEs/numerical analysis).

Example 1: Computation of fluids with phase change and convection

- Purpose: solve a very difficult systems of PDEs

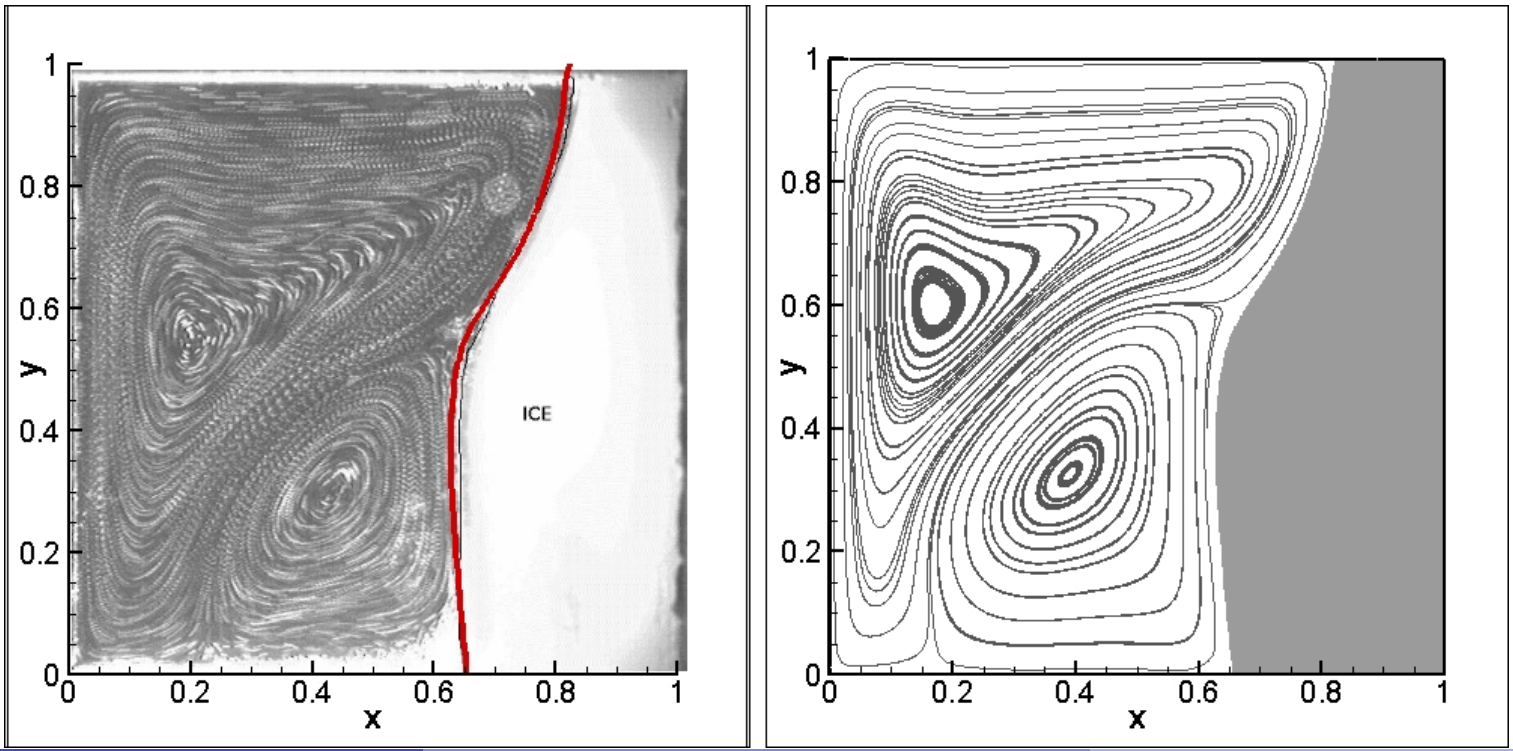
Navier-Stokes-Boussinesq equations + phase change,

- Use: classical methods $\rightarrow$ new numerical method

Taylor-Hood finite-elements + Newton method,

- I. Danaila, R. Moglan, F. Hecht, S. le Masson, JCP, 2014.

(movie) ice formation



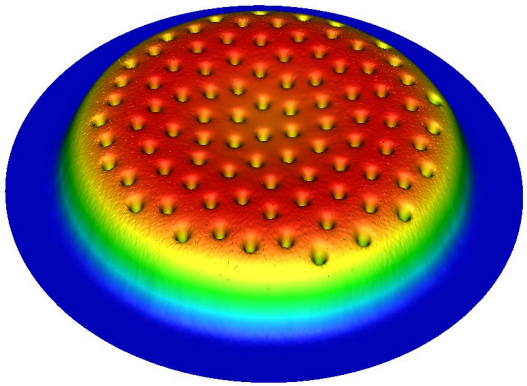
Example 2: Computation of Bose-Einstein condensates (non-linear Schrödinger equation)

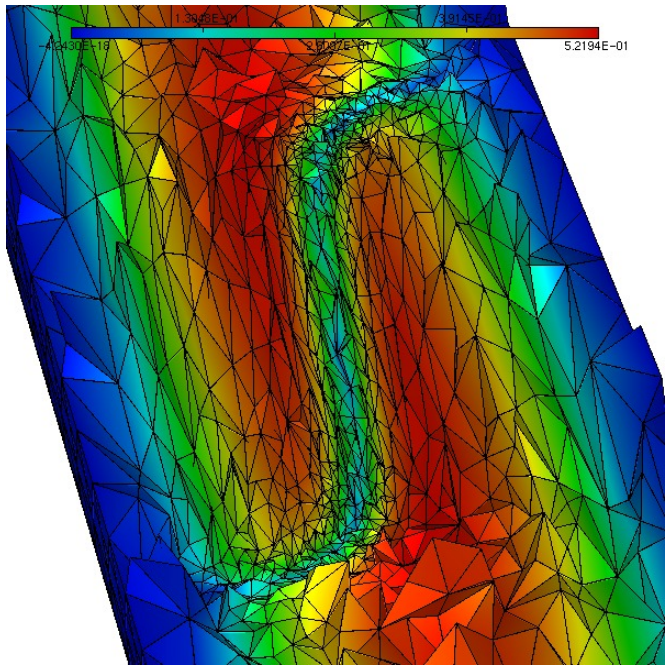
- Purpose: develop new (sophisticated) numerical algorithms

Sobolev gradient methods + Riemannian Optimization,

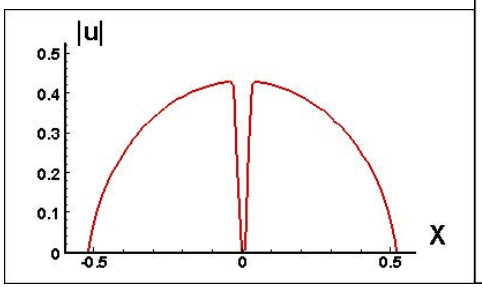
- Use: classical FE + adaptivity $\rightarrow$ new gradients, preconditioners, etc
- G. Vergez, I. Danaila, S. Auliac, F. Hecht, CPC, 2016.

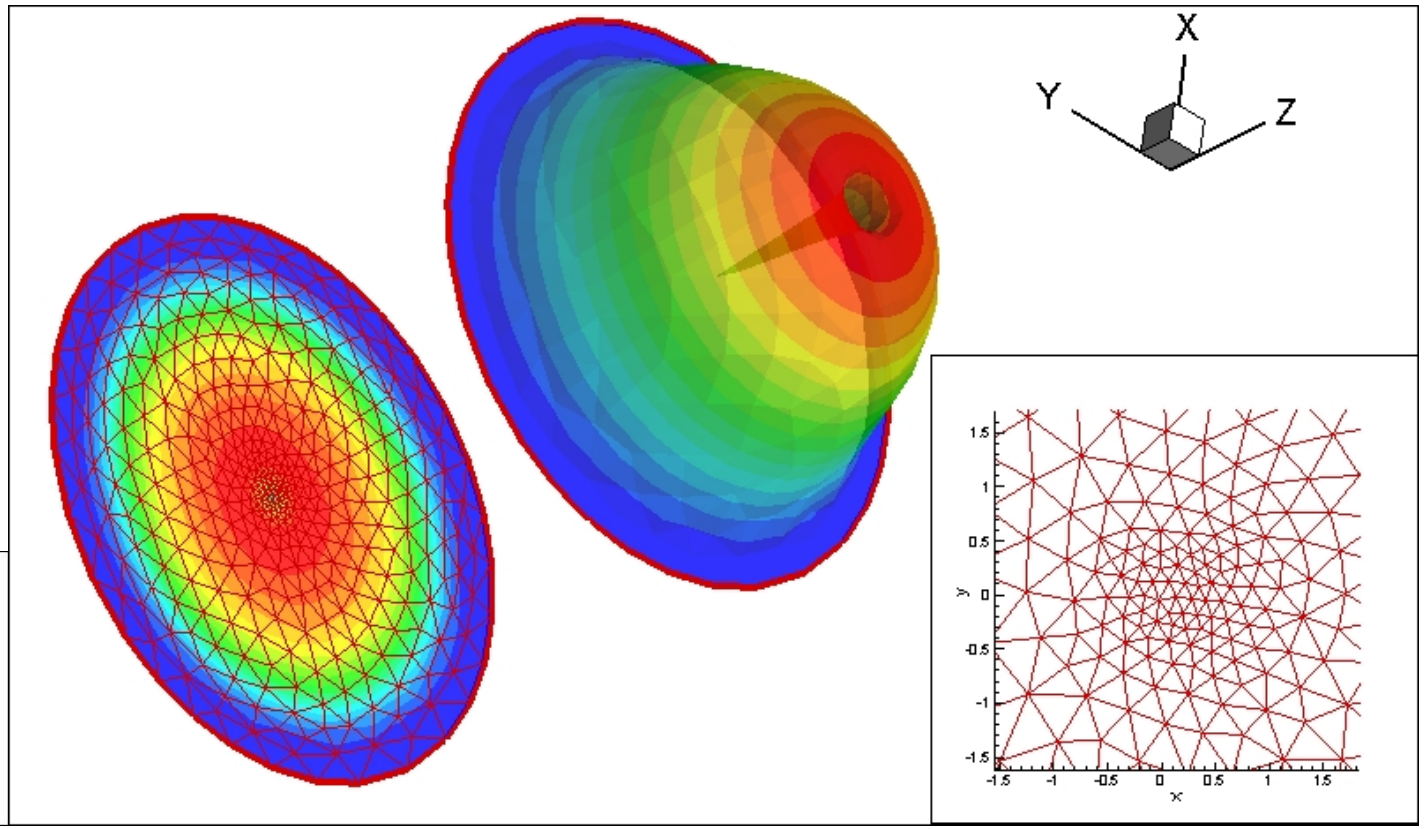
(movie) vortices inside a BEC



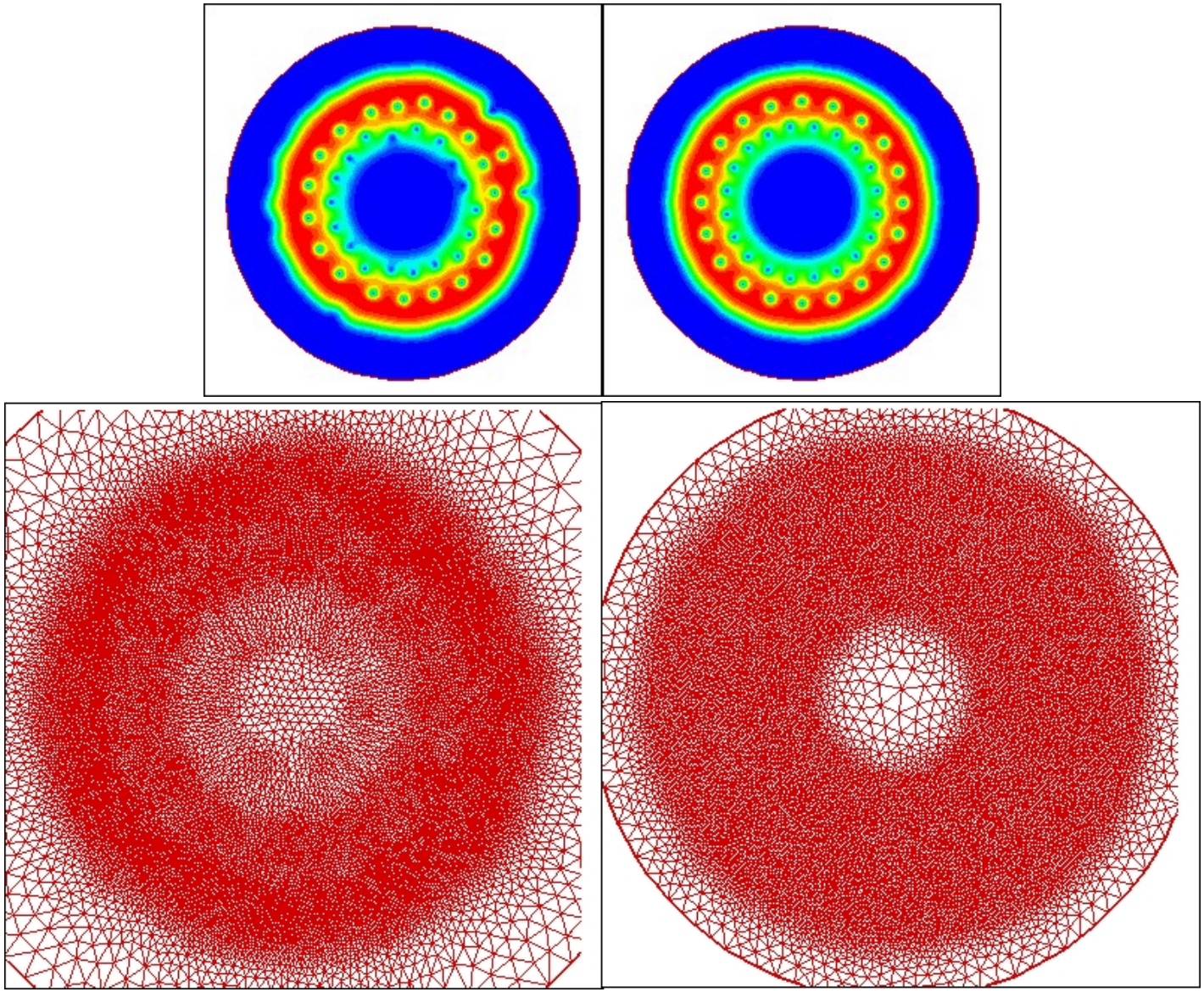


Appealing features: Mesh adaptivity

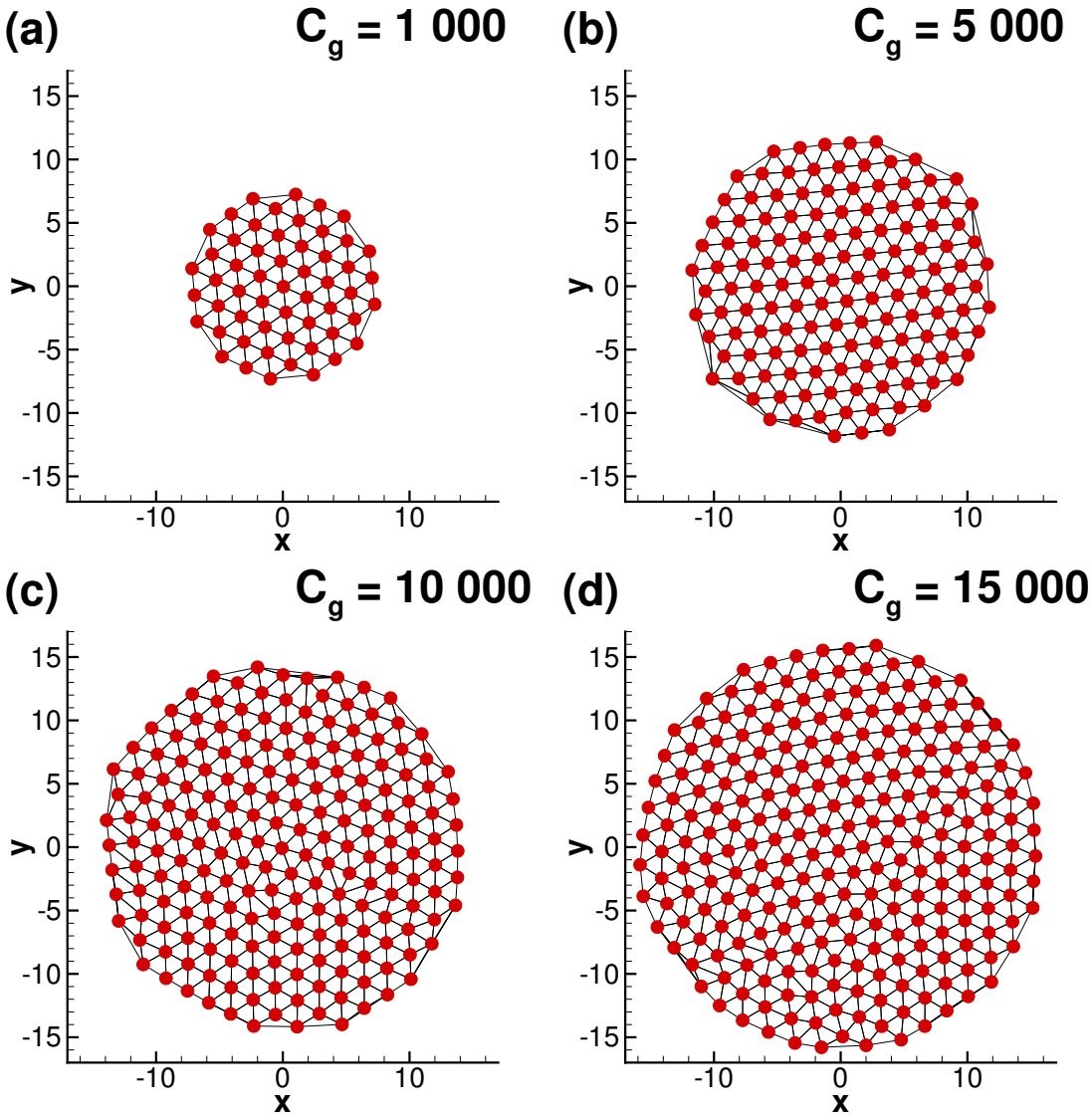




Appealing features: Controlling mesh isotropy



Appealing features: Post-processing to identify vortices, compute vortex radius, etc.



Basic features in FreeFem++

How to install and use FreeFem++?

FreeFem++: www.freefem.org or www3.freefem.org

- pre-compiled versions for Windows and MacOS,
- compilation needed for Linux,
- to write programs/scripts: use your preferred Editor (Emacs).

Explore www.freefem.org

- instructions for compilation,
- full documentation (pdf), slides from FreeFem++ days, etc.,
- Git repository (development),
- papers using FreeFem++, toolboxes, etc.,
- lots of examples (.edp scripts).
- Before starting an application: look for an example close to yours!

Basic features in FreeFem++

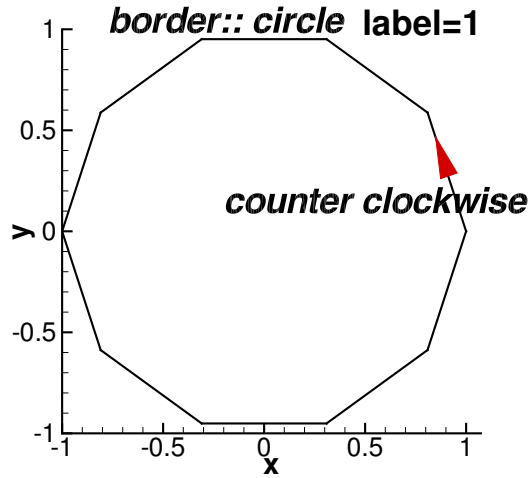
1. WHAT is FreeFem++?
2. WHY using FreeFem++?
3. **HOW to use FreeFem++ (a step-by-step guide)**
   - (1) How to install FreeFem++?
   - (2) Building a mesh
   - (3) What mathematics do you need to know?
   - (4) Solving the Poisson equation in 10 lines of code
   - (5) Dealing with Boundary Conditions
4. Summary of basic features.

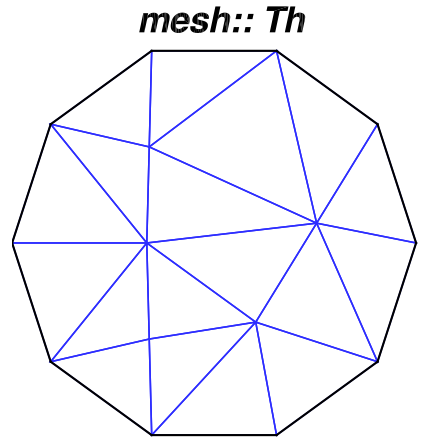
Mesh of disk (in one line of code)

Any computation starts with a mesh

mesh/mesh_circle_v01.edp

```
/* Mesh of a circle */

// Parameters

int nbseg=10;
real R=1, xc=0, yc=0;

// border
border circle(t=0,2*pi){label=1;
                        x=xc+R*cos(t);
                        y=yc+R*sin(t);}
plot(circle(nbseg),cmm="border", wait=1);

// FE mesh
mesh Th = buildmesh(circle(nbseg));
plot(Th, cmm="mesh of a circle");
```



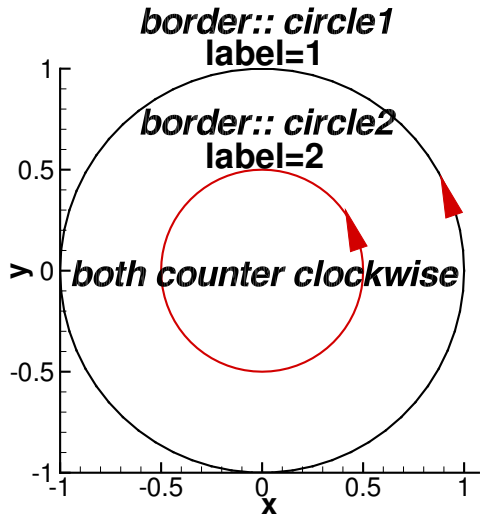


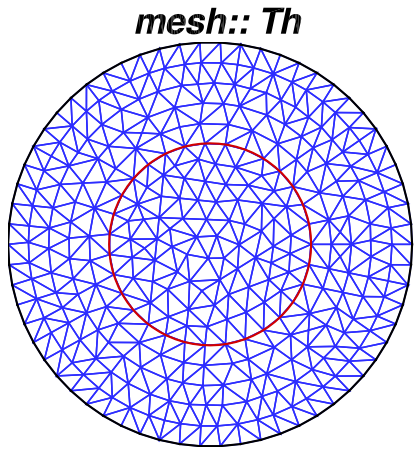
Mesh of disk (v02)

A mesh with a sub-domain:: + circle2(nbseg*2*pi*R2)

mesh/mesh_circle_v02.edp

```
int nbseg=10;real R=1, xc=0, yc=0, R2=R/2;
// borders
border circle1(t=0,2*pi){label=1;
                         x=xc+R*cos(t);
                         y=yc+R*sin(t);}
border circle2(t=0,2*pi){label=2;
                         x=xc+R2*cos(t);
                         y=yc+R2*sin(t);}
plot(circle1(nbseg*2*pi*R)+circle2(-nbseg*2*pi*R2
    ),cmm="border");


// FE mesh
mesh Th = buildmesh(circle1(nbseg*2*pi*R)
                   +circle2(-nbseg*2*pi*R2));
plot(Th, cmm="mesh of a circle with subdomain");
// Identify subdomains
cout <<"inner region:: number ="<<
  Th(xc,yc).region <<endl;
cout <<"inner region:: number ="<<
  Th(xc+(R2+R)/2,yc).region <<endl;
```



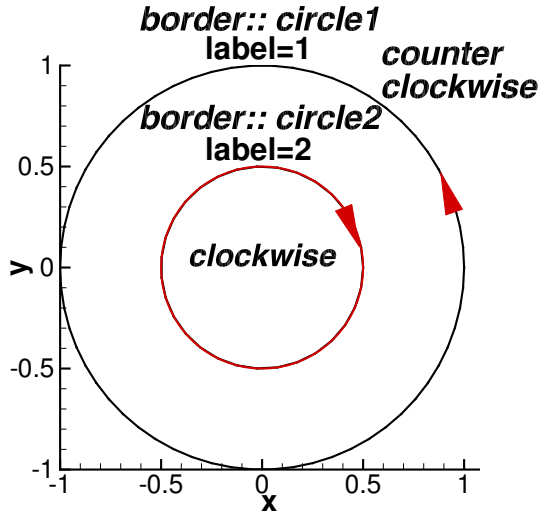


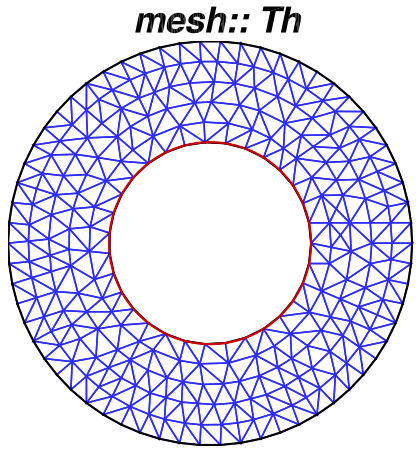
Mesh of disk (v03)

A mesh with a hole inside:: + circle2(-nbseg*2*pi*R2)

mesh/mesh_circle_v03.edp

```
/* Mesh of a circle with a hole inside */
// Parameters
int nbseg=10;
real R=1, xc=0, yc=0, R2=R/2;
// border
border circle1(t=0,2*pi){label=1;
                         x=xc+R*cos(t);
                         y=yc+R*sin(t);}
border circle2(t=0,2*pi){label=2;
                         x=xc+R2*cos(t);
                         y=yc+R2*sin(t);}
plot(circle1(nbseg*2*pi*R)+circle2(nbseg*2*pi*R2)
     ,cmm="border");
// FE mesh
mesh Th = buildmesh(circle1(nbseg*2*pi*R)
                   +circle2(-nbseg*2*pi*R2));
plot(Th, cmm="mesh of a circle with a hole");
```





Mesh of disk (v04)

A mesh with a hole inside:: using macros to avoid bugs

be carreful with the syntax of EndOfMacro and inside comments

mesh/mesh_circle_v04.edp

```
macro Bcircle(bname,Rm,xm,ym,labelm)
      /* circle border */
      border bname(t=0,2*pi)
      {label=labelm; x=xm+Rm*cos(t);y=ym+Rm*sin(t);}//EOM
// Parameters
int nbseg=10;
real R=1, xc=0, yc=0, R2=R/2;

// borders
Bcircle(circle1,R ,xc,yc,1);
Bcircle(circle2,R2,xc,yc,2);

plot(circle1(nbseg*2*pi*R)+circle2(nbseg*2*pi*R2),cmm="border");
// FE mesh
mesh Th = buildmesh(circle1(nbseg*2*pi*R)
                   +circle2(-nbseg*2*pi*R2));
plot(Th, cmm="mesh of a circle with a hole");
```

Building a mesh with FreeFem++ (v05)

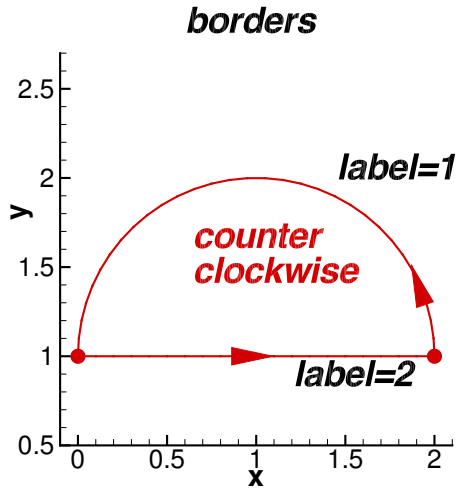
Mesh for a half-disk::

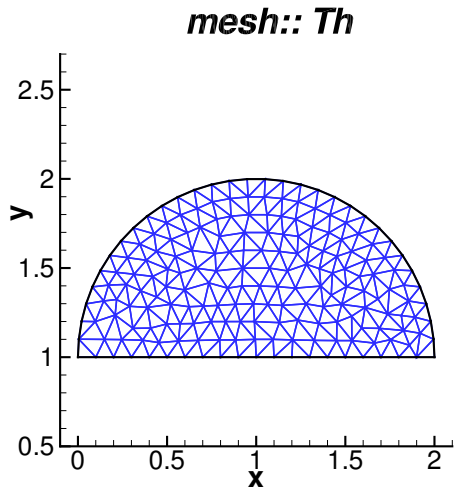
check intersection points

oriented borders (counter clockwise)

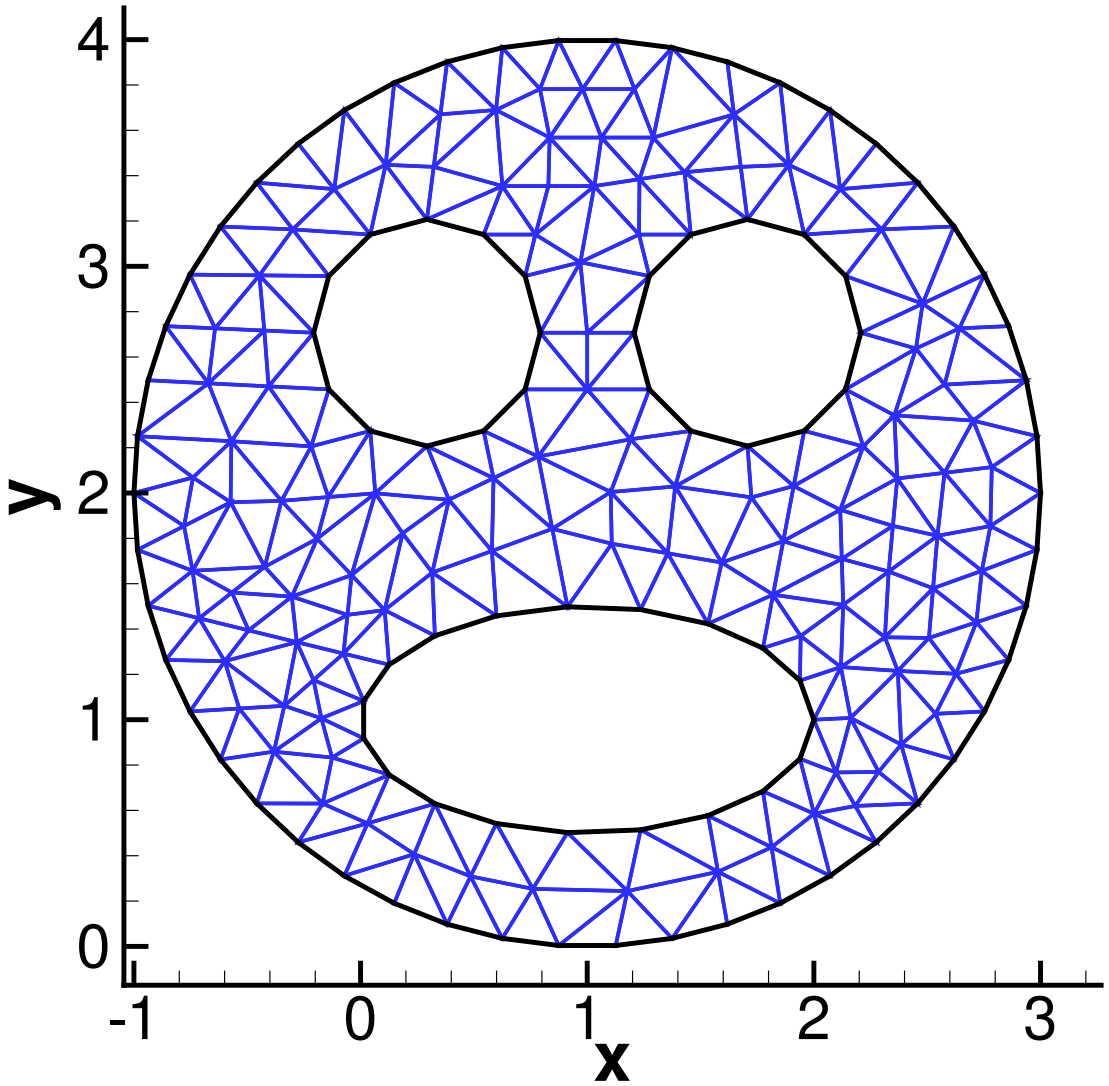
mesh_circle_v05.edp

```
/* Mesh of a half-disk  */
// Parameters
int nbseg=10;
real R=1, xc=1, yc=1;
// border
border Dcircle(t=0,pi)     {label=1;
                           x=xc+R*cos(t);
                           y=yc+R*sin(t);}
border Daxis   (t=xc-R,xc+R){label=2;
                           x=t;
                           y=yc;}
plot(Dcircle(nbseg*pi*R)+Daxis(nbseg*2*R),cmm="
    border", wait=1);
// FE mesh
mesh Th = buildmesh(Dcircle(nbseg*pi*R)+Daxis(
    nbseg*2*R));
plot(Th, cmm="mesh of a half-disk");
```





Intermission: Mesh of a smiley



Building a smiley with FreeFem++ (v06)

mesh/mesh_smiley_v01.edp

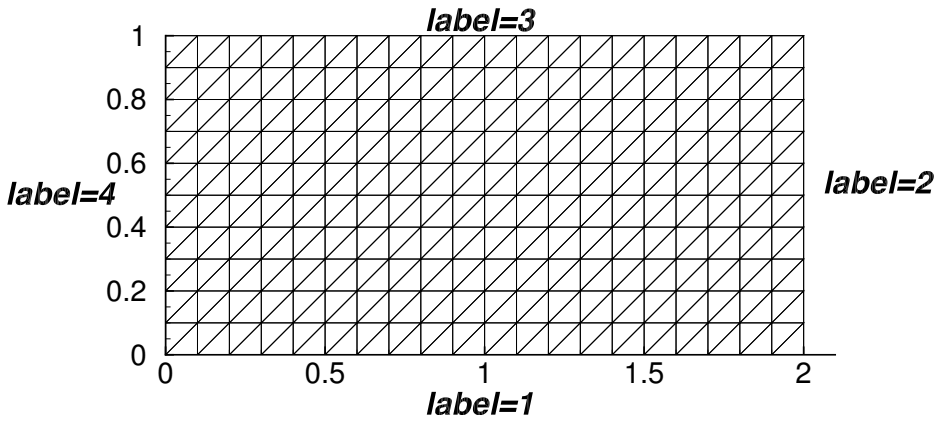
```
macro Bellipse(bname,Rmx,Rmy,xm,ym,labelm)
      border bname(t=0,2*pi)
      {label=labelm; x=xm+Rmx*cos(t);y=ym+Rmy*sin(t);}//EOM
// Parameters
int nbseg=10;
//head
real Rh=2, xh=1, yh=2, Lh=2*pi*Rh;
Bellipse(bs1,Rh,Rh,xh,yh,1);
//eyes
real xy1=xh+Rh/2*cos(pi/4), yy=yh+Rh/2*sin(pi/4), Ry=Rh/4, Ly=2*pi*Ry;
Bellipse(bs2,Ry,Ry,xy1,yy,2);
real xy2=xh-Rh/2*cos(pi/4);
Bellipse(bs3,Ry,Ry,xy2,yy,3);
//mouth
real a=Rh/2, b=Rh/4, Lm=pi*sqrt(2*(a^2+b^2));
Bellipse(bs4,a,b,xh+0,yh-Rh/2,4);

plot(bs1(nbseg*Lh)+bs2(nbseg*Ly)+bs3(nbseg*Ly)+bs4(nbseg*Lm));
// FE mesh
mesh Th = buildmesh(bs1(nbseg*Lh)+bs2(10*nbseg*Ly)+bs3(-nbseg*Ly)+bs4(-
    nbseg*Lm));
plot(Th, cmm="mesh of a smiley");
```

Mesh of a rectangle (in one line of code)

Mesh a rectangle using the built-in function ”square”

mesh/mesh_rectangle_v01.edp

```
/* Mesh of a rectangle using square
    function  */
// Parameters
int nbseg=10;
real L=2,H=1;
real xc1=0,    yc1=0;
// FE mesh
mesh Th = square(nbseg*L,
                 nbseg*H,
          [xc1+x*L,yc1+y*H]);
plot(Th, cmm="mesh of a rectangle");
```

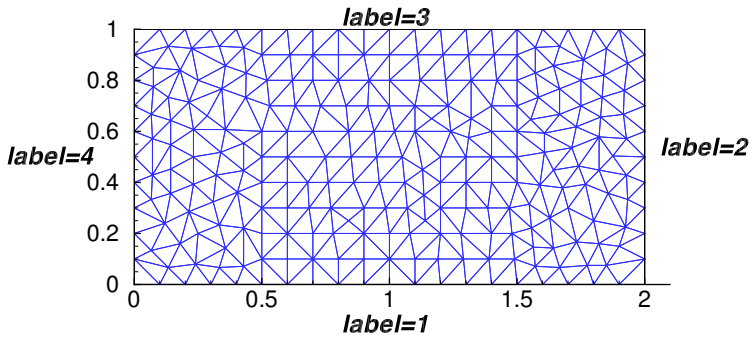


Mesh of a rectangle (building each border)

mesh/mesh_rectangle_v02.edp

```
macro Bsegment(bname,xP1,yP1,xP2,yP2,Ls,labelm)
      real Ls=sqrt((xP1-xP2)^2+(yP1-yP2)^2);
      border bname(t=0,Ls)
      {label=labelm; x=xP1+t*(xP2-xP1)/Ls; y=yP1+t*(yP2-yP1)/Ls;}//EOM
// Parameters
```

```
real L=2,H=1;
real xc1=0,     yc1=0,
     xc2=xc1+L,yc2=yc1,
     xc3=xc2,   yc3=yc2+H,
     xc4=xc1,   yc4=yc3+L;
//borders
Bsegment(bs1,xc1,yc1,xc2,yc2,Ls1,1);
Bsegment(bs2,xc2,yc2,xc3,yc3,Ls2,2);
Bsegment(bs3,xc3,yc3,xc4,yc4,Ls3,3);
Bsegment(bs4,xc4,yc4,xc1,yc1,Ls4,4);

mesh Th = buildmesh(bs1(nbseg*Ls1)+bs2(
    nbseg*Ls2)+bs3(nbseg*Ls3)+bs4(nbseg*
    Ls4));
plot(Th, cmm="mesh of a rectangle");
```

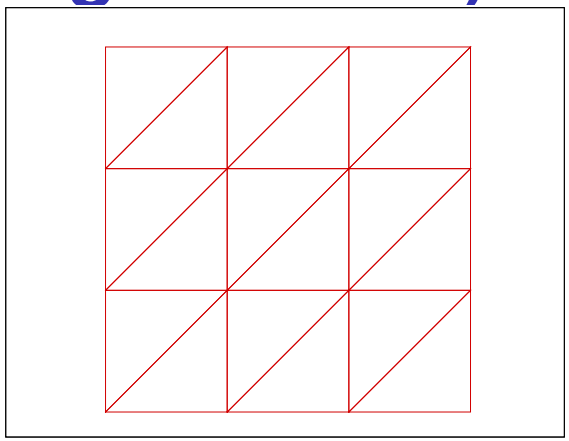


Basic features in FreeFem++

Finite element representation (Lagrange P^1 here)

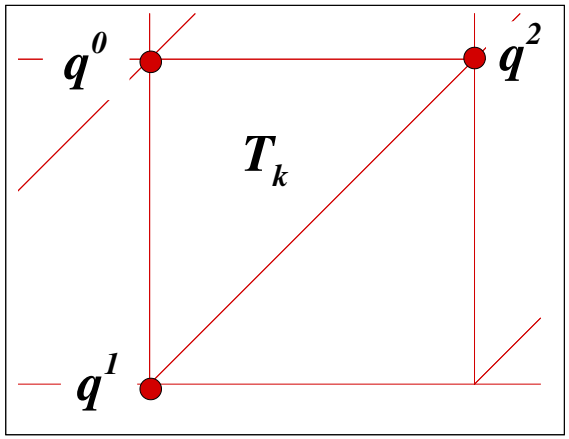
- **Functional (Sobolev) spaces**

$$H^1(\Omega) = \{v \in L^2(\Omega) : \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y} \in L^2(\Omega)\}$$
$$V(\Omega) = \{v \in H^1(\Omega) : v|_{\Gamma^D} = 0\}.$$



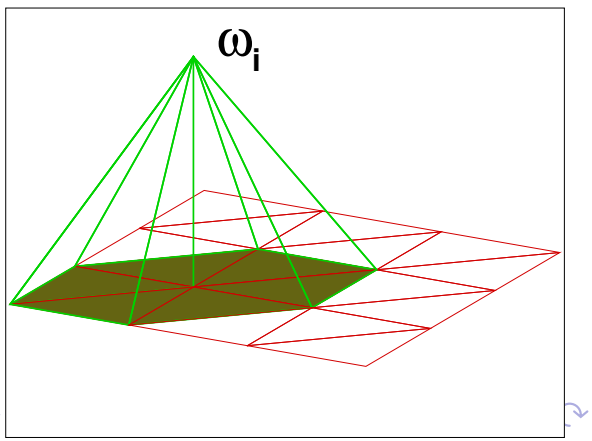
- **Approximation spaces**

$\mathcal{T}_h$ ::= triangulation,
$\Omega_h = \cup_{k=1}^{n_t} T_k$, ($n_t$ is the number of triangles).
$H_h = \{v \in C^0(\Omega_h) : \forall T_k \in \mathcal{T}_h, v|_{T_k} \in P^1(T_k)\}$,
$V_h = \{v \in H_h : v|_{\Gamma_h^D} = 0\}$.



- **Basis functions**

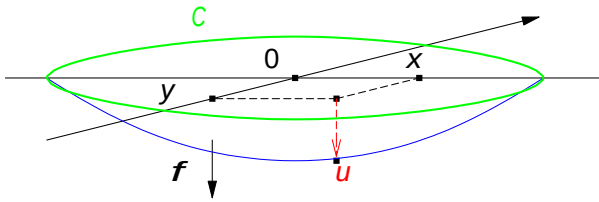
$w^i \in H_h$, $w^i(q^j) = \delta_{ij}$ (1 if $i = j$, 0 otherwise).
$\nabla w^i|_{T_k} = const$,
$dim(H_h) = n_v$ (n_v is the number of vertices),
$f_h \in H_h$ ::= array of n_v values.



Weak (variational) formulations (the Poisson equation here)

- **Deformation of a circular membrane**

$$\begin{cases} -\Delta u = f & \text{for} \quad (x,y) \in \mathcal{D} \\ u = 0 & \text{for} \quad (x,y) \in \partial\mathcal{D} = \mathcal{C} \end{cases}$$



- **Variational (weak) formulation**:

– multiply by a test function

$$v \in V(\mathcal{D}) = \{v \in H^1(\mathcal{D}) \ : \ v|_{\mathcal{C}} = 0\}$$

– use Green's formula (integration by parts)

$$\int_{\mathcal{D}} [-v\Delta u] \, dxdy = \int_{\mathcal{D}} \nabla v \nabla u - \int_{\mathcal{C}} \frac{\partial u}{\partial n} v \, d\gamma,$$

– to obtain (notice that $v|_{\mathcal{C}} = 0$)

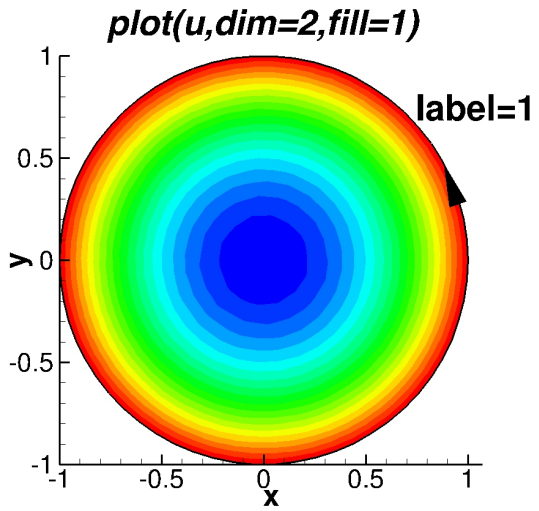
$$\int_{\mathcal{D}} \nabla v \nabla u - \int_{\mathcal{D}} fv = 0,$$

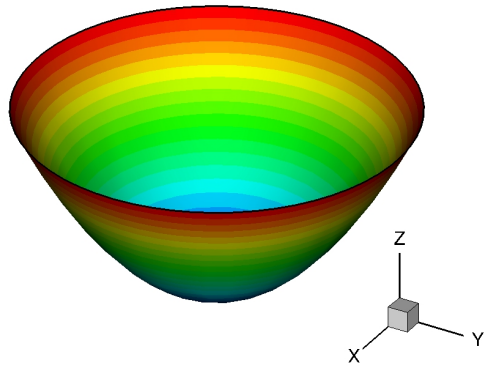
Definition of finite element spaces, integrals

FEspace is a type (as real or int) in FreeFem++

fespace_v01.edp

```
/* P1 representation */
// Parameters
int nbseg=200;
real R=1, xc=0, yc=0;
// border
border circle(t=0,2*pi){label=1;x=xc+R*cos(t);
                                y=yc+R*sin(t);}
// FE mesh
mesh Th = buildmesh(circle(nbseg));plot(Th);
// FE space
fespace Vh(Th, P2);
Vh u=x*x+y*y;  plot(u, dim=3,fill=1,wait=1);


// 2d integral
cout.precision(12);
real int2dU = int2d(Th)(u);
cout <<"int2dU="<<int2dU<<endl;
cout <<" exact="<<pi*R^4/2<<endl;
real int1dU = int1d(Th,1)(u);
cout <<"int1dU="<<int1dU<<endl;
```





plot(u,dim=3,fill=1)

Functions: x, y, z are reserved names for space variables

(part of) fespace_v02.edp

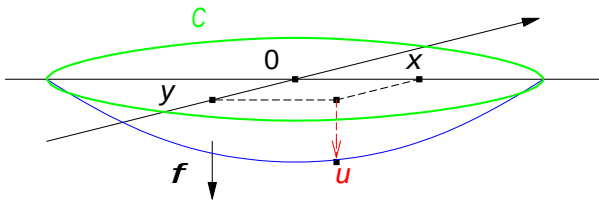
```
func fu=x*x+y*y;
cout.precision(12);
real int2dU = int2d(Th)(fu);
cout <<"int2dU="<<int2dU<<endl;
cout <<" exact="<<pi*R^4/2<<endl;
real int1dU = int1d(Th,1)(fu);
cout <<"int1dU="<<int1dU<<endl;
cout <<" exact="<<2*pi*R^3<<endl;
// other definition of functions
func string infoMesh(mesh & Th)
{
  string message="Mesh size:: nv="+ Th.nv+" nt="+Th.nt;
      return message;
}
cout << infoMesh(Th)<<endl;
func  real Meshsize(mesh & Th)
{
  fespace Ph(Th,P0);
  Ph h=hTriangle;
      return h[].max;
}
cout << Meshsize(Th)<<endl;
```

Basic features in FreeFem++

The Poisson equation (1)

- **Deformation of a circular membrane**

$$\begin{cases} -\Delta u = f & \text{for} \quad (x,y) \in \mathcal{D} \\ u = 0 & \text{for} \quad (x,y) \in \partial\mathcal{D} = \mathcal{C} \end{cases}$$



- **Variational (weak) formulation**:

– multiply by a test function

$$v \in V(\mathcal{D}) = \{v \in H^1(\mathcal{D}) \ : \ v|_{\mathcal{C}} = 0\}$$

– use Green's formula (integration by parts)

$$\int_{\mathcal{D}} [-v\Delta u] \, dxdy = \int_{\mathcal{D}} \nabla v \nabla u - \int_{\mathcal{C}} \frac{\partial u}{\partial n} v \, d\gamma,$$

– to obtain (notice that $v|_{\mathcal{C}} = 0$)

$$\int_{\mathcal{D}} \nabla v \nabla u - \int_{\mathcal{D}} fv = 0,$$

The Poisson equation (2)

$$\int_{\mathcal{D}} \nabla v \nabla u - \int_{\mathcal{D}} fv = 0 \Leftrightarrow \int_{\mathcal{D}} \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \right) - \int_{\mathcal{D}} fv = 0$$

- derive the discrete weak formulation: FreeFem++ will take care!

(part of) lap/lap_v01.edp

```
mesh Th = buildmesh(circle(nbseg));plot(Th);

// Data of the problem
func fs=4;  // RHS (source) function


// FE space
fespace Vh(Th, P1);
// Variational (weak formulation)
Vh u,v;     // u=unknown, v=test function
```

The Poisson equation (3)

- using a macro for the gradient = column array of two componets $grad(u) = \left[\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right]$ and $grad(u)'$ is the transposed gradient (row array)

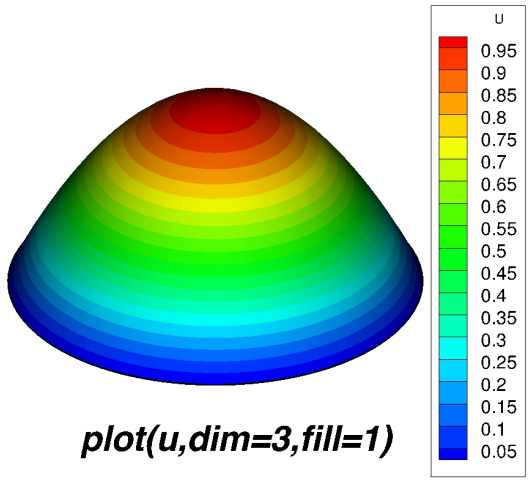
(part of) lap/lap_v01b.edp

```
Vh u,v;     // u=unknown, v=test function
Vh uexact=R^2-x^2-y^2;//exact solution


macro grad(u) [dx(u), dy(u)]//EOM


  problem Poisson(u,v)=int2d(Th)(grad(u)'*grad(v))
                      -int2d(Th)(fs*v)
                      +on(1,u=0); // Dirichlet bc
// Solve the problem, plot the solution
```

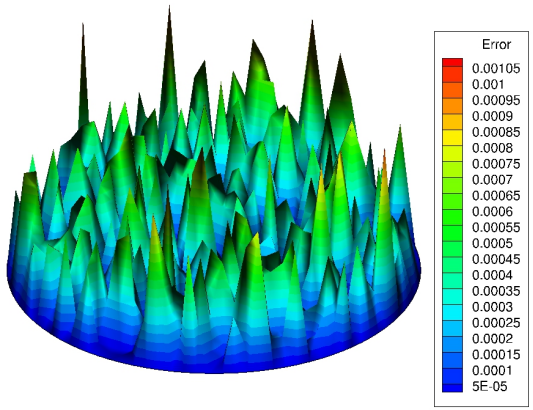
FreeFem++ program for the Poisson equation

lap/lap_v01b.edp

```
int nbseg=100;real R=1, xc=0, yc=0;
border circle(t=0,2*pi){label=1;x=xc+R*cos(t);
                                  y=yc+R*sin(t);}
mesh Th = buildmesh(circle(nbseg));plot(Th);
// Data of the problem
func fs=4;  // RHS (source) function
// FE space
fespace Vh(Th, P1);
// Variational (weak formulation)
Vh u,v;     // u=unknown, v=test function
Vh uexact=R^2-x^2-y^2;//exact solution


macro grad(u) [dx(u), dy(u)]//EOM


  problem Poisson(u,v)=int2d(Th)(grad(u)'*grad(v)
      )
                    -int2d(Th)(fs*v)
                    +on(1,u=0); // Dirichlet bc
// Solve the problem, plot the solution
Poisson; plot(u,dim=2,fill=1);
// Compare with the exact solution
```



plot(u,dim=3,fill=1)



plot(error,dim=3,fill=1)

Versatility of the software: change the accuracy

- to switch from P1 to P2 just change the definition of the FE-space (for P3 and P4 also load the corresponding module)

(part of) lap/lap_v01c.edp

```
// FE space
fespace Vh(Th, P2);
```

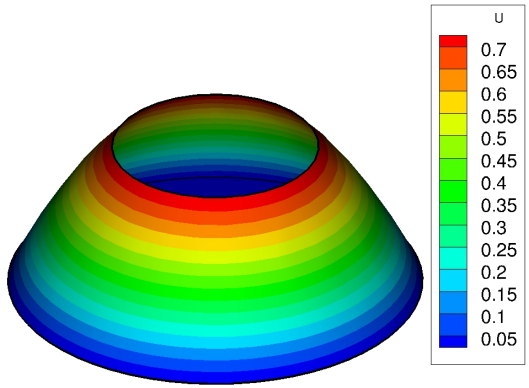
(part of) lap/lap_v01d.edp

```
// FE space
load "Element_P3";
fespace Vh(Th, P3);
```

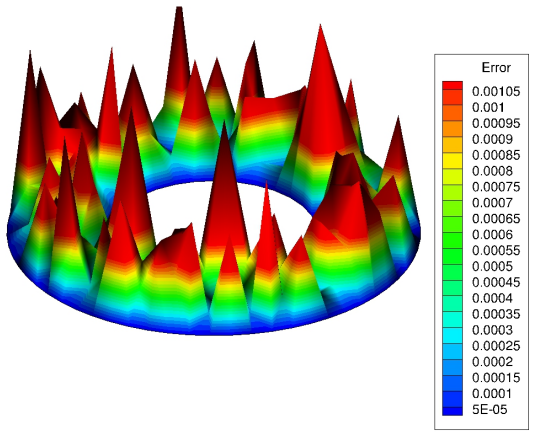
Versatility of the software: change the mesh

lap/lap_v02.edp

```
include "../mesh/mesh_circle_v03.edp";

// Data of the problem
func fs=4;  // RHS (source) function
// FE space
fespace Vh(Th, P1);
// Variational (weak formulation)
Vh u,v;     // u=unknown, v=test function
Vh uexact=R^2-x^2-y^2;//exact solution
macro grad(u) [dx(u), dy(u)]//EOM
problem Poisson(u,v)=int2d(Th)(grad(u)'*grad(v))
                    -int2d(Th)(fs*v)
                    +on(1,2, u=uexact); // exact
                        Dirichlet bc
// Solve the problem, plot the solution
Poisson; plot(u,dim=2,fill=1);
// Compare with the exact solution
Vh error=abs(u-uexact);
plot(error,dim=3,fill=1);
cout.precision(12);
cout<<"Maximum error ="<<error[].linfty<<endl;
cout<<"Maximum error ="<<error[].max<<endl;
```



plot(u,dim=3,fill=1)



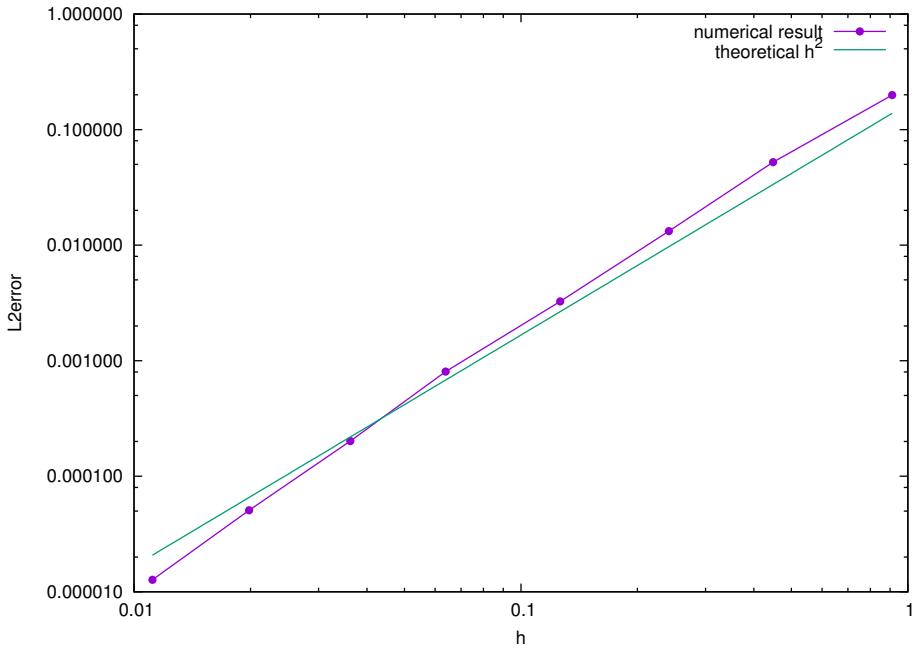
plot(error,dim=3,fill=1)

Convergence of the FE method

$$\|u - u_{ex}\|_2 = \sqrt{\int_\Omega (u - u_{ex})^2}$$

$$\|u - u_{ex}\|_2 \leq Ch^2$$

Exercise: Implement the H^1 norm or semi-norm!



Convergence of the FE method (script part1)

(part 1 of) lap_v02conv.edp

```
/* Convergence of the P1 FEM */
int nbseg=5;real R=1, xc=0, yc=0;
border circle(t=0,2*pi){label=1;x=xc+R*cos(t);
                                y=yc+R*sin(t);}
mesh Th = buildmesh(circle(nbseg));
fespace Vh(Th, P1);
func fs =4;  // RHS (source) function
func fex=R^2-x^2-y^2;

func  real Meshsize(mesh & Th)
{
  fespace Ph(Th,P0);
  Ph h=hTriangle;
      return h[].max;
}

Vh u,v,uexact,error;     // u=unknown, v=test function
macro grad(u) [dx(u), dy(u)]//EOM
problem Poisson(u,v) =    int2d(Th)(grad(u)'*grad(v))
  - int2d(Th)(fs*v)+ on(1,u=0); // Dirichlet bc
```

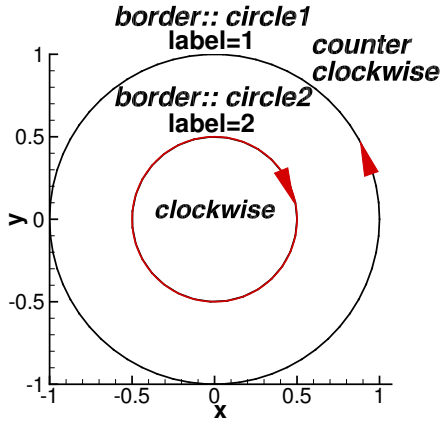
Convergence of the FE method (script part 2)

(part 2 of) lap_v02conv.edp

```
{verbosity=0; // lowest verbosity mode (no internal cout)
ofstream ferror("lap_v02_error.dat");
 real eL2, h, eL2old=1,hold=1;
for(int irun=0;irun<8;irun++)
  {
    nbseg *= 2;
    Th=buildmesh(circle(nbseg));
     Poisson;
    eL2 = sqrt(int2d(Th)(square(u-fex)));
    h   = Meshsize(Th);
    real order =log(eL2/eL2old)/log(h/hold);
    eL2old=eL2;
    hold  =h;
      cout<<"nbseg= "<<nbseg<<"  Mesh size= "<<h <<"   nv= "<<Th.nv
      <<"   L2error= "<<eL2<<"  order= "<<order<<endl;
    ferror<<h <<"  "<<eL2<<"  "<< order<<endl;

  }
}
```

Basic features in FreeFem++

Boundary conditions (1)

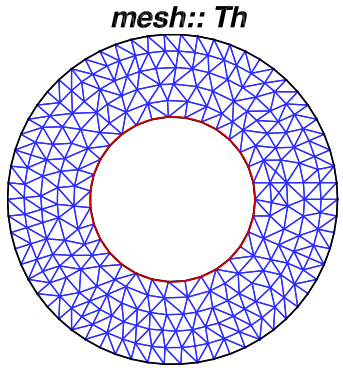


- Consider the Poisson (heat) equation:

$$-\Delta u = f, \quad \text{in} \quad \Omega$$

- with boundary conditions:

$$\begin{cases} \text{on } \Gamma_2 & u = u_{hot} & \text{Dirichlet BC} \\ \text{on } \Gamma_1 & \dfrac{\partial u}{\partial n} + \alpha u = 0 & \text{Neumann/Fourier BC} \\ & (\alpha \geq 0) \end{cases}$$



- Weak formulation:

$$\int_\Omega [-v\Delta u] = \int_\Omega \nabla v \nabla u - \int_\Gamma \frac{\partial u}{\partial n} v,$$

$$\int_\Omega \nabla v \nabla u - \int_\Omega f v - \int_\Gamma \frac{\partial u}{\partial n} v = 0.$$

Boundary conditions (2)

$$\int_\Omega \nabla v \nabla u - \int_\Omega fv - \sum_{i=1}^{2} \int_{\Gamma_i} \frac{\partial u}{\partial n} v = 0.$$

```
Vh u,v;     // u=unknown, v=test function
problem Poisson(u,v) =int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
                      -int2d(Th)(fs*v)
```

$$\begin{cases} \text{on } \Gamma_2 \quad u = u_{hot} \qquad \text{Dirichlet} \quad v = 0 \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} v = 0 \\ +on(2, u = uhot) \\ \text{on } \Gamma_1 \quad \frac{\partial u}{\partial n} + \alpha u = 0 \quad \text{Fourier} \quad \frac{\partial u}{\partial n} = -\alpha u \qquad \int_{\Gamma_1} \frac{\partial u}{\partial n} v = \int_{\Gamma_1} (-\alpha u v) \\ +int1d(Th,1)(alpha * u * v); \end{cases}$$

Warning: important to carefully identify the borders

- for homogeneous Neumann BC ($\alpha = 0$) $\Longrightarrow$ nothing to be specified;
- if for a border nothing is specified (with "on" or "int1d") $\Longrightarrow$ homogeneous Neumann BC is implicitly imposed!

Boundary conditions (2)

$$\int_\Omega \nabla v \nabla u - \int_\Omega fv - \sum_{i=1}^{2} \int_{\Gamma_i} \frac{\partial u}{\partial n} v = 0.$$

```
Vh u,v;     // u=unknown, v=test function
problem Poisson(u,v) =int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
                      -int2d(Th)(fs*v)
```

$$\begin{cases} \text{on } \Gamma_2 \quad u = u_{hot} \quad \text{Dirichlet} \quad v = 0 \quad \int_{\Gamma_2} \frac{\partial u}{\partial n} v = 0 \\ +on(2, u = uhot) \\ \text{on } \Gamma_1 \quad \frac{\partial u}{\partial n} + \alpha u = 0 \quad \text{Fourier} \quad \frac{\partial u}{\partial n} = -\alpha u \quad \int_{\Gamma_1} \frac{\partial u}{\partial n} v = \int_{\Gamma_1} (-\alpha u v) \\ +int1d(Th,1)(alpha * u * v); \end{cases}$$

Warning: important to carefully identify the borders

- for homogeneous Neumann BC ($\alpha = 0$) $\Longrightarrow$ nothing to be specified;
- if for a border nothing is specified (with "on" or "int1d") $\Longrightarrow$ homogeneous Neumann BC is implicitly imposed!

Boundary conditions (3): the script

lap/lap_v03.edp

```
include "../mesh/mesh_circle_v03.edp";

// Data of the problem
func fs=0;  // RHS (source) function
// FE space
fespace Vh(Th, P1);
// Variational (weak formulation)
Vh u,v;     // u=unknown, v=test function

real uhot=10;
real alpha=10;
macro grad(u) [dx(u), dy(u)]//EOM
problem Poisson(u,v)=int2d(Th)(grad(u)'*grad(v))
                     -int2d(Th)(fs*v)
                     +int1d(Th,1)(alpha*u*v)//from Fourier bc
                     +on(2, u=uhot); // Dirichlet bc
// Solve the problem, plot the solution
Poisson; plot(u,dim=3,fill=1,value=1);

cout<<"Maximum value ="<<u[].max<<endl;
cout<<"Minimum value ="<<u[].min<<endl;
```

Exercises for the Poisson equation (1)

In the previous problem (see lap_v01.edp): we chose $f = 4$ corresponding to $u_{ex}(x, y) = R^2 - x^2 - y^2$ which satisfies the BC.

1. Solve the same problem on the half disk. What BC should be applied at the axis ($y = 0$)? How this new BC appears in the variational formulation?
2. Solve the following PDE

$$-\Delta u + u = f \quad \text{in} \quad \mathcal{R}, \quad \text{and} \quad u|_{\partial\mathcal{R}} = g,$$

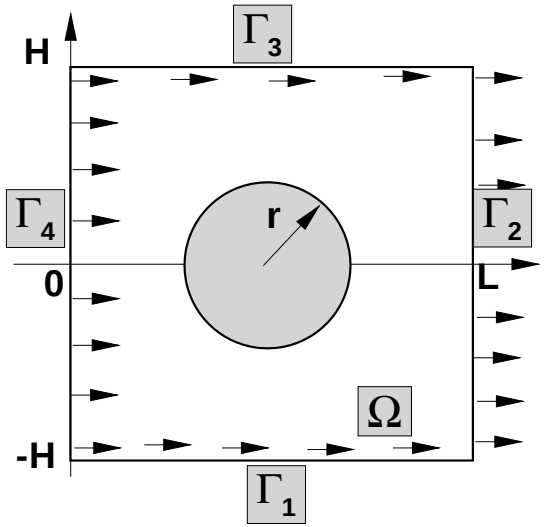
where $\mathcal{R} = [0, 1] \times [0, 1]$ is a square domain. Find the expression for $f(x, y)$ and $g(x, y)$ such as that the exact solution should be $u_{ex}(x, y) = 1 - x^2 - y^2$.

Exercises for the Poisson equation (2)

3. Potential flow around a cylinder (solution script: lap_ex3.edp).

For each point (x, y) the flow velocity is derived from the potential φ

$$\vec{v} = \nabla\varphi = \begin{pmatrix} \frac{\partial\varphi}{\partial x} \\ \frac{\partial\varphi}{\partial y} \end{pmatrix}, \quad -\Delta\varphi = 0.$$



$\Omega = [0, L] \times [-H, H]$.
The cylinder of radius r is centred in the domain.

on $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$: $\vec{v} = v_0\,\vec{e}_x$, where $\vec{e}_x$ is the unit vetor alog x-axis.

on the cylinder Γ_5 : $\vec{v} \cdot \vec{n} = 0$, zero normal velocity.

Summary of basic features (1)

Basic C++ syntax + FE layer (meta-language)

- **real** (double precision), **integer**, **bool**, **string** ;
- arrays (**real [int] v(n);**) , full matrices (**real [int, int] A(n,n);**) ;
- **if** clauses, **for** loops, etc.

Summary of basic features (1)

Basic C++ syntax + FE layer (meta-language)

- **real** (double precision), **integer**, **bool**, **string** ;
- arrays (**real [int] v(n);**) , full matrices (**real [int, int] A(n,n);**) ;
- **if** clauses, **for** loops, etc.

mesh Th=buildmesh(...)

- Automatic Delaunay triangulation (2D)/ **tetgen** in 3D;
- possibility to save the mesh (savemesh);
- possibility to load a mesh (generated by another software, **Gmsh**).

Summary of basic features (1)

Basic C++ syntax + FE layer (meta-language)

- **real** (double precision), **integer**, **bool**, **string** ;
- arrays (**real [int] v(n);**) , full matrices (**real [int, int] A(n,n);**) ;
- **if** clauses, **for** loops, etc.

mesh Th=buildmesh(...)

- Automatic Delaunay triangulation (2D)/ **tetgen** in 3D;
- possibility to save the mesh (savemesh);
- possibility to load a mesh (generated by another software, **Gmsh**).

fespace Vh(Th, P1)

- **Vh** is a type (like **integer** or **real**);
- definition of FE-variables **Vh u, v;**;
- the only line to change if other FE is needed: **fespace Vh(Th, P2)**.

Summary of basic features (2)

Basic brick: transcription of the weak formulation
problem Lap(u,v)= int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)) - int2d(Th)(fs*v) + on(1,u=g);

- **u** is the unknown, **v** the test function
- **int2d(Th)(...)** $\Longleftrightarrow \int_{\mathcal{D}}(...)$
- **int1d(Th,2)(...)** $\Longleftrightarrow \int_{\Gamma_2}(...)$
- **on(1, u=g)** $\Longleftrightarrow$ on Γ_2, $u = g(x, y)$
- **dx(u)** $\Longleftrightarrow \partial u/\partial x$
- $+$ other operators related to FE (normal vector, etc).

Summary of basic features (3)

What's behind the scene? FreeFem++

- identifies **u** as the unknown, **v** as the test function,
- identifies the bilinear form
$\mathcal{A}(u, v) = int2d(Th)(dx(u) * dx(v) + dy(u) * dy(v))$,
- creates the associated sparse matrix **A** for the declared FE-type,
- identifies the linear form $\ell(v) = int2d(Th)(fs * v)$,
- creates **b** the associated RHS for the declared FE-type,
- includes Dirichlet BC from **on** instructions (penalisation of **A** and **b**),
- solves the linear system $\mathbf{AU} = \mathbf{b}$,
- identifies u with the array **U** (switch between representations).

Summary of basic features (3)

What's behind the scene? FreeFem++

- identifies **u** as the unknown, **v** as the test function,
- identifies the bilinear form
$\mathcal{A}(u, v) = int2d(Th)(dx(u) * dx(v) + dy(u) * dy(v))$,
- creates the associated sparse matrix **A** for the declared FE-type,
- identifies the linear form $\ell(v) = int2d(Th)(fs * v)$,
- creates **b** the associated RHS for the declared FE-type,
- includes Dirichlet BC from **on** instructions (penalisation of **A** and **b**),
- solves the linear system $\mathbf{AU} = \mathbf{b}$,
- identifies u with the array **U** (switch between representations).

Good news

- FreeFem++ take care of all FE-technicalities,
- the **problem** formulation is symbolic, evaluated when called,
(no need to rewrite the **problem** if any changes, in f or g, etc.),
- the user can control the process: create separately the arrays **A**, **b**,
select the linear solver, impose BC, etc.