Finite Element Modelling with FreeFem++
Part I: Basic features in FreeFem++

Ionut Danaila[1],

[1]Université de Rouen Normandie, France

University of Strathclyde, 30th of June, 2017.

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) What mathematics do you need to know?
   - (3) Building a mesh
   - (4) Solving the Poisson equation in 10 lines of code
   - (5) Dealing with Boundary Conditions
4. **Summary of basic features.**
   - Summary of basic features
5. **Towards advanced features**
   - From steady to time-dependent PDEs
   - Build FE-matrices
   - Mesh adaptivity

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, 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 (theory of PDEs/numerical analysis) theories,

Example 1: Computation of fluids with phase change and convection

- Purpose: solve a very difficult systems of PDEs

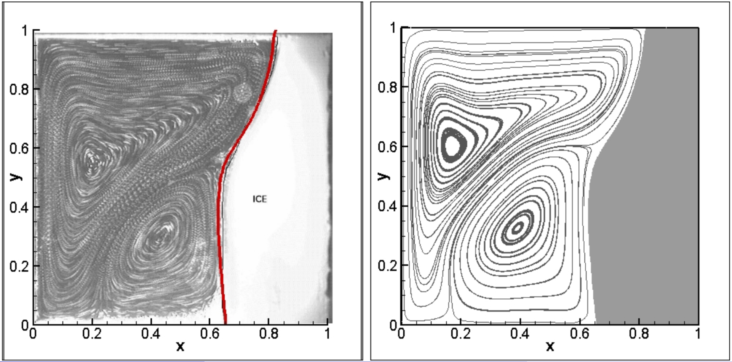
Navier-Stokes-Boussinesq equations + phase change,

- 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



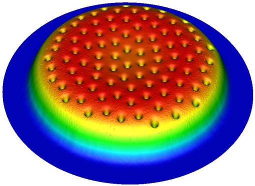
Example 2: Computation of Bose-Einstein condensates (non-linear Schrödinger equation)

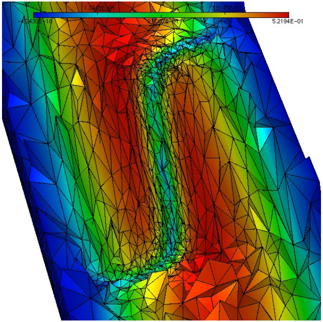
- Purpose: develop new (sophisticated) numerical algorithms

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





Basic features in FreeFem++

How to install and use FreeFem++?

FreeFem++: www.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, slides from FreeFem++ days, etc
- lots of examples (.edp scripts).

FreeFem++-js: https://www.ljll.math.upmc.fr/~lehyaric/ffjs

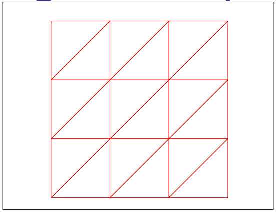
- Run FreeFem++ scripts online.

Basic features in FreeFem++

Finite element representation (Lagrange P^1 here)

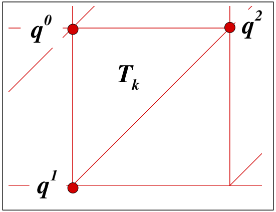
- **Functional (Sobolev) spaces**

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



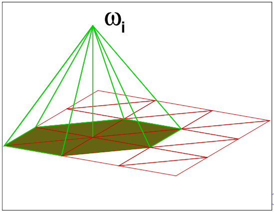
- **Approximation spaces**

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



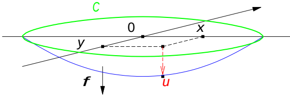
- **Basis functions**

$w^i \in H_h, \quad w^i(q^j) = \delta_{ij}$ (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 $v|_{\mathcal{C}} = 0$)

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

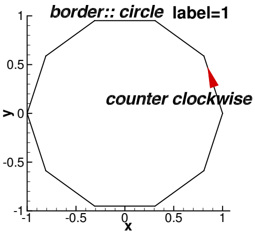
Basic features in FreeFem++

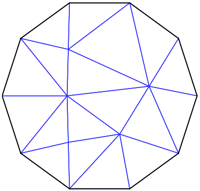
Mesh of disk (in one line of code)

Any computation starts with a mesh

mesh/mesh_circle_v01.edp

```
/* Mesh of a circle */

// Parameters

int nbseg=100;
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");

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



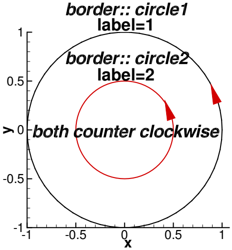


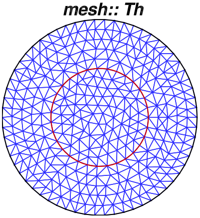
Mesh of disk (v02)

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=2*pi,0){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;
```



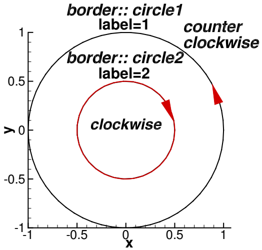


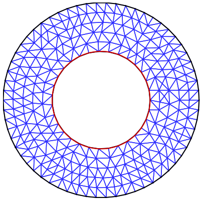
Mesh of disk (v03)

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");
```





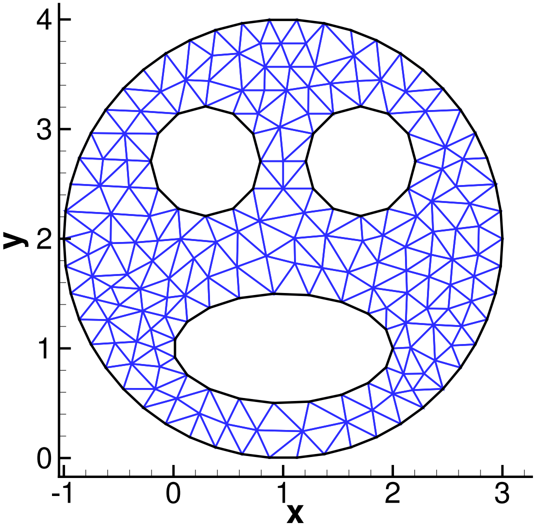
Mesh of disk (v04)

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");
```

Intermission: Mesh of a smiley



Building a smiley with FreeFem++ (v06)

mesh/mesh_smiley_v01.edp

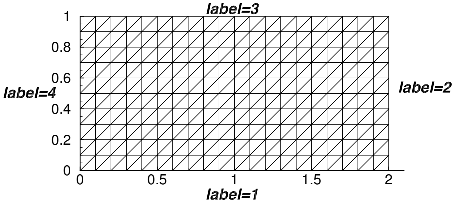
```
macro Bellipse(bname,Rmx,Rmy,xm,ym,labelm)
      border bname(t=0,2*pi)
      {label=labelm; x=xm+Rmx*cos(t);y=ym+Rmy*sin(t);}//EOM
// Parameters
int nbseg=10;
//head
real Rh=2, xh=1, yh=2, Lh=2*pi*Rh;
Bellipse(bs1,Rh,Rh,xh,yh,1);
//eyes
real xy1=xh+Rh/2*cos(pi/4), yy=yh+Rh/2*sin(pi/4), Ry=Rh/4, 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");
```

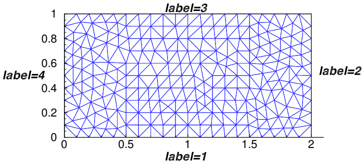


Mesh of a rectangle (building each border)

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");
```

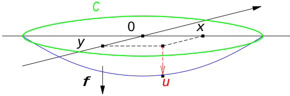


Basic features in FreeFem++

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

```
// 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
problem Poisson(u,v) =
  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
  - int2d(Th)(fs*v)
  + on(1,u=0); // Dirichlet boundary condition
// Solve the problem, plot the solution
```

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)

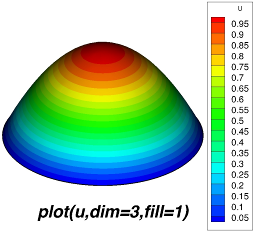
(part of) lap/lap_v01b.edp

```
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
Poisson; plot(u,dim=2,fill=1);
// Compare with the exact solution
Vh error=abs(u-uexact);
plot(error,dim=3,fill=1);
```

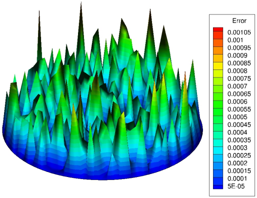
FreeFem++ program for the Poisson equation

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
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)

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);
```

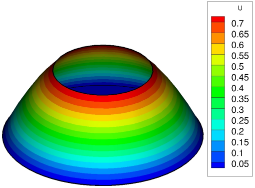
(part of) lap/lap_v01d.edp

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

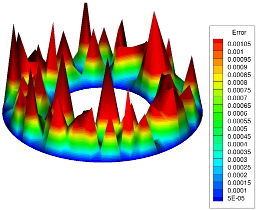
Versatility of the software: change the mesh

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)

Basic features in FreeFem++

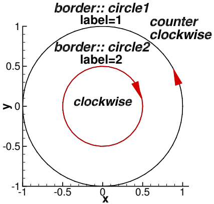
Boundary conditions (1)

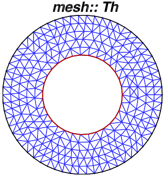
- Consider the Poisson (heat) equation:

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

- 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)

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

```
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)
```

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

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)

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

```
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)
```

on Γ_2 $\quad u = u_{hot}$ $\quad$ Dirichlet $\quad v = 0$ $\quad \int_{\Gamma_2} \frac{\partial u}{\partial n} v = 0$

$+on(2, u = uhot)$

on Γ_1 $\quad \frac{\partial u}{\partial n} + \alpha u = 0$ $\quad$ Fourier $\quad \frac{\partial u}{\partial n} = -\alpha u$ $\quad \int_{\Gamma_1} \frac{\partial u}{\partial n} v = \int_{\Gamma_1} (-\alpha u v)$

$+int1d(Th, 1)(alpha * u * v);$

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;
```

Basic features in FreeFem++

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)(...)** $\Longleftrightarrow \int_{\mathcal{D}}(\ldots)$
- **int1d(Th,2)(...)** $\Longleftrightarrow \int_{\Gamma_2}(\ldots)$
- **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.

Basic features in FreeFem++

Solving the time-dependent heat equation (1)

$$\frac{\partial \theta}{\partial t} - \Delta\theta = 0, \quad \text{for} \quad (x, y) \in \Omega, \quad 0 \le t \le t_{max}$$

$$+\text{Boundary Conditions(in space)} + \text{Initial Condition}(t = 0).$$

- Discretisation in time (FD finite-difference type)

$$[0, t_{max}] = \bigcup_{n=0}^{N-2} [t_n, t_n + \delta t], \ t_n = n\delta t, \ n = 0, 1, \ldots, N-1, \ \delta t = T/(N-1).$$

Notation $\theta^n(x) = \theta(x, t_n)$.

$$\frac{\theta^{n+1}(x) - \theta^n(x)}{\delta t} - \Delta\theta^{n+1}(x) = 0 \quad \text{(implicit scheme)}$$

$$\frac{\theta^{n+1}(x) - \theta^n(x)}{\delta t} - \Delta\theta^{n}(x) = 0 \quad \text{(explicit scheme)}$$

Solving the time-dependent heat equation (2)

- Discretisation in space (FE finite-element type): implicit scheme

$$\int_\Omega \frac{\theta^{n+1}}{\delta t} v - \int_\Omega \frac{\theta^n}{\delta t} v + \int_\Omega \left[-v \Delta \theta^{n+1} \right] = 0$$

$$\int_\Omega \frac{\theta^{n+1}}{\delta t} v - \int_\Omega \frac{\theta^n}{\delta t} v + \int_\Omega \nabla \theta^{n+1} \nabla v - \int_\Gamma \frac{\partial \theta^{n+1}}{\partial n} v = 0$$

- Weak formulation ready to use with FreeFem++: impose (spatial) BC on θ^{n+1} as for the stationary problem.
- In programs, in the "time loop" we use only two variables: $u = \theta^{n+1}$ and $uold = \theta^n$.

Script for the time-dependent heat equation (1)

$$\int_\Omega \frac{u}{\delta t} v - \int_\Omega \frac{uold}{\delta t} v + \int_\Omega \nabla u \nabla v - \int_\Gamma \frac{\partial u}{\partial n} v = 0$$

(part 1 of) time-dep/heat_time_v01.edp

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

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

real uhot=10, alpha=10;

//Time-evolution formulation
real tmax=0.1, dt=0.001, idt=1./dt;
Vh uold=0;
macro grad(u) [dx(u), dy(u)]//EOM
problem HeatTime(u,v)=int2d(Th)(idt*u*v)-int2d(Th)(idt*uold*v)
                    +int2d(Th)(grad(u)'*grad(v))
                    +int1d(Th,1)(alpha*u*v)//from Fourier bc
                    +on(2, u=uhot); // Dirichlet bc
```

Script for the time-dependent heat equation (2)

(part 2 of) time-dep/heat_time_v01.edp

```
//Time loop
real t=0; verbosity=0;
while (t <= tmax)
{
   t+=dt;
   HeatTime;
   plot(u,dim=3,cmm="Time t="+t,fill=1);
   cout<<"Time="<< t<<"  Max(u) ="<<u[].max<<"  Min(u) ="<<u[].min<<
       endl;
uold=u;
}
```

Basic features in FreeFem++

Time-dependent heat equation with matrices (1)

$$\int_\Omega \frac{u}{\delta t}v - \int_\Omega \frac{uold}{\delta t}v + \int_\Omega \nabla u \nabla v - \int_\Gamma \frac{\partial u}{\partial n}v = 0$$

$$\mathcal{A}(u,v) = \ell(v)$$

$$\mathcal{A}(u,v) = \int_\Omega \frac{uv}{\delta t} + \int_\Omega \nabla u \nabla v - \int_\Gamma \frac{\partial u}{\partial n}v \Longrightarrow (matrix)\ \mathbf{A}$$

$$\ell(v) = \int_\Omega \frac{uold}{\delta t}v \Longrightarrow (rhs)\ \mathbf{b} = \mathbf{A}_{mass} * \mathbf{uold}$$

$$\mathcal{A}_{mass}(u,v) = \int_\Omega \frac{uv}{\delta t}$$

+ impose Dirichlet BC by penalisation (tgv technique)

Time-dependent heat equation with matrices (2)

(part of) time-dep/heat_time_v02.edp

```
//------------------- matrix of the system
   real tgv=1e30;
   varf   Vsys(u,v) =  int2d(Th)(idt*u*v)
                      +int2d(Th)(grad(u)'*grad(v))
                      +int1d(Th,1)(alpha*u*v)
                      + on(2,u=uhot);  // + on(2,u=1); the same matrix

   matrix Asys      =  Vsys(Vh,Vh,tgv=tgv);

//------------------- Mass matrix
   varf  Vmass(u,v) = int2d(Th)(u*v*idt) ;
   matrix Amass      =  Vmass(Vh,Vh,tgv=tgv);

//------------------- right-hand side term + (boundary conditions)
   Vh BC;
   varf  Vbc(u,v) = on (2,u=uhot);
   BC[] = Vbc(0,Vh,tgv=tgv);
```

Time-dependent heat equation with matrices (3)

(part of) time-dep/heat_time_v02.edp

```
// BC0 = 0 for nodes on Gamma2, BC0=1 elsewhere

   Vh BC0;
   BC0[] = Vbc(0,Vh,tgv=1);// BC0=1 for nodes on Gamma2, BC=0 elsewhere
   BC0   = -BC0;
   BC0[]  +=1;    //now BC0 = 0 for nodes on Gamma2, BC0=1 elsewhere
```

Time-dependent heat equation with matrices (4)

(part of) time-dep/heat_time_v02.edp

```
//Time loop
real t=0; verbosity=0;
real [int] rhs = BC[]; // fix the correct dimension

   set(Asys,solver=UMFPACK);

while (t <= tmax)
{
   t+=dt;

// prepare the rhs
  rhs    = Amass*uold[];
  rhs .*= BC0[];     // set to zero the value for nodes on Gamma2
  rhs   += BC[];     // set the correct value on Gamma2

// solve the linear system
 u[]= Asys^-1*rhs;

   plot(u,dim=3,cmm="Time t="+t,fill=1);
   cout<<"Time="<< t<<"  Max(u) ="<<u[].max<<"  Min(u) ="<<u[].min<<
       endl;
uold=u;
}
```

Basic features in FreeFem++

Time-dependent heat equation with mesh adaptivity

(part of) time-dep/heat_time_v03.edp

```
//Time loop
real t=0; verbosity=0;
real errorAdapt=0.01;
while (t <= tmax)
{
   t+=dt;
   HeatTime;
   plot(Th,u,dim=2,cmm="Time t="+t,fill=0);
   cout<<"Time="<< t<<"  Max(u) ="<<u[].max<<"  Min(u) ="<<u[].min<<
       endl;
Th=adaptmesh(Th,u,uold,inquire=1,err=errorAdapt,iso=1);
u=u;
uold=u;
}
```