Solving PDEs with finite elements: an introduction to free software FreeFem++ Part II: Solving PDEs

Ionut Danaila

University of Rouen Normandie, France

ionut.danaila.perso.math.cnrs.fr

Institute of Mathematical Sciences, Singapore.
November 14, 2019.

Soving PDEs with FreeFem++

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

1. **Towards advanced features**
   - From steady to time-dependent PDEs
   - Build FE-matrices
   - Mesh adaptivity
2. **Linear elasticity problems**
   - Linear Lamé equations
   - Simplified model for a dam
   - Deformation of a beam
   - Vibration of a beam
3. **Solving non-linear problems**
   - From steady to time-dependent PDEs
4. **Summary of Part II**

Basic features in FreeFem++

Solving the time-dependent heat equation (1)

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

$+$Boundary Conditions(in space) $+$ 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/heat_time_v01.edp

```
include "../../Part_01_Scripts/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/heat_time_v01.edp

```
//Time loop
real t=0; verbosity=0;
while (t <= tmax)
{
    t+=dt;
    HeatTime;
    plot(u,dim=2,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/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/heat_time_v02.edp

```
//------------------- array for boundary conditions
// BC0 = 0 for nodes on Gamma2, BC0=1 elsewhere

   varf  Vbc0(u,v) = on (2,u=1);
   Vh BC0;
   BC0[] = Vbc0(0,Vh,tgv=1);// BC0=1 for nodes on Gamma2, 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/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/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);
u=u;
uold=u;
}
```

Solving the wave equation (1)

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = 0, \quad \text{for} \quad (x, y) \in \Omega, \quad 0 \leq t \leq t_{max}$$

$+$Boundary Conditions(in space) $+$ Initial Condition for u, $\frac{\partial u}{\partial t}(t = 0)$.

- 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 $u^n(x) = u(x, t_n)$.

$$\frac{u^{n+1}(x) - 2u^n(x) + u^{n-1}(x)}{\delta t^2} - \Delta u^n(x) = 0.$$

Solving the wave equation (2)

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

$$\int_\Omega \frac{u^{n+1}}{\delta t} v - \int_\Omega \frac{2u^n}{\delta t} v + \int_\Omega \frac{u^{n-1}}{\delta t} v + \int_\Omega \left[-v \Delta u^n \right] = 0$$

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

- Weak formulation ready to use with FreeFem++: impose BC on u^n as for the steady problem.
- In programs, in the "time loop" we use only three variables: $u = u^{n+1}$, $uold = u^n$, $uvold = u^{n-1}$.

Script for the wave equation (1)

(part 1 of) smiley_heat_D.edp

```
/* Wave equation on a Smiley   */

include "../../Part_01_Scripts/mesh/mesh_smiley_v01.edp";

// New FE mesh (close the eyes and the mouth)
Th = buildmesh(bs1(nbseg*Lh)+bs2(-nbseg*Ly)+bs3(nbseg*Ly)
           +bs4(nbseg*Lm));plot(Th, cmm="New mesh of a smiley");

//Time-evolution data
real tmax=5, dt=0.01, idt2=1./(dt*dt);
// FE space
fespace Vh(Th, P1);

// Variational (weak formulation)
Vh u,v, uold=0,uvold=0;
macro grad(u) [dx(u), dy(u)]//EOM
problem SmileyW(u,v)=int2d(Th)(idt2*u*v)-int2d(Th)(2*idt2*uold*v)+int2d
     (Th)(idt2*uvold*v)
+int2d(Th)(grad(uold)'*grad(v))
; // only Neumann boundary conditions
```

Script for the wave equation (2)

(part 2 of) smiley_heat_D.edp

```
//Time loop
real t=0; int iter=0, nplot=2;
// Initial condition (u is Gaussian and du/dt=0)
   uvold = exp(-2*((x-xh)^2 + (y-yh)^2));
   uold  = uvold;
   u     = uold;

verbosity=0;
while (t <= tmax)
{
   iter++;t+=dt;SmileyW;
   if(!(iter%nplot))// visualisation every nplot
     {
     plot(u,cmm="Wave t="+t,fill=1,dim=2);
     cout <<"t="<<t<<"  u min= "<< u[].min
                    <<"  u max="<< u[].max <<endl;
     }
uvold=uold;
uold =u;
}
```

Basic features in FreeFem++

Linear elasticity (1)

- Let us denote by $\vec{f}$ the external force field acting on a solid Ω.

Let $\vec{u} = (X - x, Y - y) = (u_1, u_2)$ the displacement vector of (x, y).
Let $\vec{\vec{\sigma}}(\vec{u}) = (\sigma_{ij}(\vec{u}))$ the tensor of constraints,
and $\vec{\vec{\epsilon}}(\vec{u}) = \epsilon_{ij}(\vec{u})$ the deformation tensor:

- For small displacements $\vec{u}$ and an elastic solid, the Hook law:

$$\sigma_{ij}(u) = \lambda \delta_{ij} \nabla . \vec{u} + 2\mu \epsilon_{ij}(\vec{u}), \quad \vec{\vec{\sigma}} = \lambda (\nabla . \vec{u}) \vec{\vec{I}} + 2\mu \vec{\vec{\epsilon}}. \tag{1}$$

with δ_{ij} the Kronecker symbol, and

$$\epsilon_{ij}(u) = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad \vec{\vec{\epsilon}}(\vec{u}) = \frac{1}{2} \left(\vec{\vec{\nabla}} \vec{u} + \vec{\vec{\nabla}}^T \vec{u} \right). \tag{2}$$

The constants λ, μ (Lamé coefficients) are expressed as a function of E (elasticity modulus) and ν (Young modulus)

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{3}$$

Linear elasticity (2)

- Lamé system of equations for linear elasticity:

$$-\mu\Delta\vec{u} - (\mu+\lambda)\nabla(\nabla.\vec{u}) = \vec{f} \text{ dans } \Omega, \tag{4}$$

- To write the weak formulation, we use the equilibrium between efforts and constraints:

$$-div(\vec{\vec{\sigma}}) = \vec{f}, \quad \text{or} \quad \begin{cases} \dfrac{\partial}{\partial x}\sigma_{11} + \dfrac{\partial}{\partial y}\sigma_{12} + f_1 = 0, \\ \dfrac{\partial}{\partial x}\sigma_{12} + \dfrac{\partial}{\partial y}\sigma_{22} + f_2 = 0. \end{cases} \tag{5}$$

We multiply the previous system with a vector test function $\vec{v} = (v_1, v_2)$ and integrate over Ω. After integration by parts:

$$\int_\Omega \left(\sigma_{11}\frac{\partial v_1}{\partial x} + \sigma_{12}\frac{\partial v_1}{\partial y}\right) + \left(\sigma_{12}\frac{\partial v_2}{\partial x} + \sigma_{22}\frac{\partial v_2}{\partial y}\right) - \int_\Omega (f_1 v_1 + f_2 v_2)$$

$$- \int_{\partial\Omega} (\sigma_{11}n_1 + \sigma_{12}n_2)\, v_1 - \int_{\partial\Omega} (\sigma_{12}n_1 + \sigma_{22}n_2)\, v_2 = 0 \tag{6}$$

Linear elasticity (3)

or:

$$\int_\Omega \lambda\,(\nabla.\vec{u})(\nabla.\vec{v}) + \int_\Omega 2\mu\,\vec{\vec{\epsilon}}(\vec{u}) : \vec{\vec{\epsilon}}(\vec{v}) - \int_\Omega \vec{f}.\vec{v} - \int_{\partial\Omega} \left(\vec{\vec{\sigma}}.\vec{n}\right).\vec{v} = 0, \quad (7)$$

with $\vec{\vec{a}} : \vec{\vec{b}} = \sum_{i,j} a_{ij} b_{ij}$ the tensor contraction (inner product).

Astutely compute the contraction of the tensors:

$$\vec{\vec{\epsilon}}(\vec{u}) : \vec{\vec{\epsilon}}(\vec{v}) = \frac{\partial u_1}{\partial x}\frac{\partial v_1}{\partial x} + \frac{\partial u_2}{\partial y}\frac{\partial v_2}{\partial y} + \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right)\left(\frac{\partial v_1}{\partial y} + \frac{\partial v_2}{\partial x}\right) \quad (8)$$

$$\vec{\vec{\epsilon}}(\vec{u}) : \vec{\vec{\epsilon}}(\vec{v}) = \left[\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial y}, \frac{1}{\sqrt{2}}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right)\right]' * \left[\frac{\partial v_1}{\partial x}, \frac{\partial v_2}{\partial y}, \frac{1}{\sqrt{2}}\left(\frac{\partial v_1}{\partial y} + \frac{\partial v_2}{\partial x}\right)\right] \quad (9)$$

Basic features in FreeFem++

Simplified model for a dam (1)

Boundary conditions

The dam is (hopefully) well anchored on the ground:

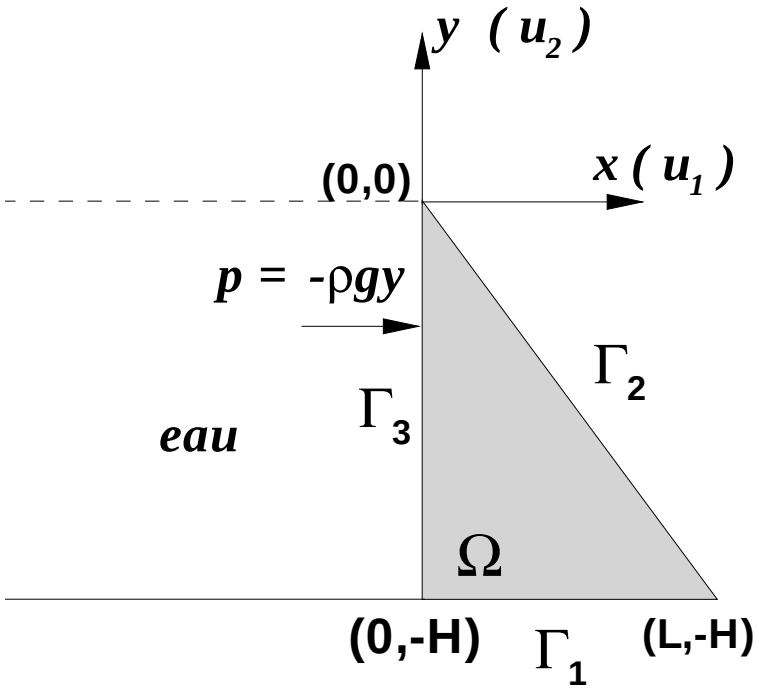
$$u_1 = u_2 = 0, \quad \text{sur} \quad \Gamma_1.$$

On the right side Γ_2 the constraint is null:

$$\vec{\vec{\sigma}}.\vec{n} = 0, \quad \text{on} \quad \Gamma_2.$$

On the left side Γ_3 the water pressure, $p = -\rho g y$, acts perpendicularly to the surface of the dam:

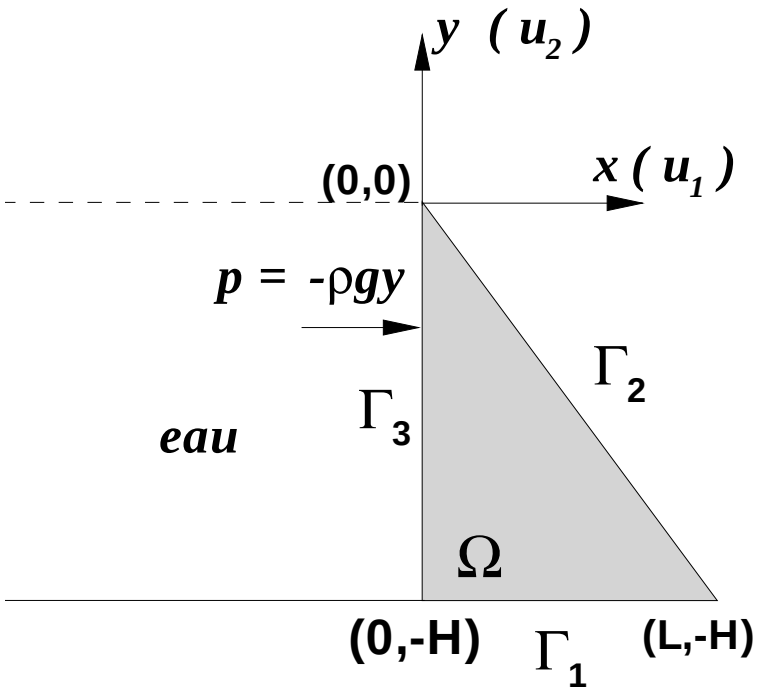
$$\vec{\vec{\sigma}}.\vec{n} = p\vec{e}_x, \quad \text{on} \quad \Gamma_3.$$



Simplified model for a dam (2)

elast/dam_v01.edp

```
// Build the mesh
//============================
real L=1, H=1;

border Gamma1(t=0,L)   {x=t;   y=-H;}
                  //
    down
real L1=L;
border Gamma2(t=L,0)   {x=t;   y=-H*(x/L); }
      //
    right
real L2=sqrt(L*L+H*H);
border Gamma3(t=0,-H)   {x=0;   y=t ;}
               // left (in contact with
    water)
real L3=H;

int nbseg=10;
mesh Th = buildmesh( Gamma1(nbseg*L1) +
    Gamma2(nbseg*L3) + Gamma3(nbseg*L3) );
```



Simplified model for a dam (3)

elast/dam_v01.edp

```
// Vectorial FE spaces
//============================

fespace Vh1(Th,P1);
fespace Vh(Th,[P1, P1]);
Vh [u1,u2], [v1,v2];
Vh1 p = -0.05*y;

// Problem
//============================

problem pb([u1,u2],[v1,v2]) =
int2d(Th)(
  2.*mu*(dx(u1)*dx(v1)+dy(u2)*dy(v2)
  +0.5*(dx(u2)+dy(u1))*(dx(v2)+dy(v1)))
  +lambda*(dx(u1)+dy(u2))*(dx(v1)+dy(v2))
)
- int1d(Th, Gamma3)(p*v1)
+ on(Gamma1, u1=0, u2=0);
```

Simplified model for a dam (4)

elast/dam_v01.edp

```
// Solution
//============================

pb;
plot(u1, value=1, wait=1, cmm="u1: Deformation following x");
plot(u2, value=1, wait=1, cmm="u2: Deformation following y");
plot( [u1,u2], wait=1, cmm="Deformation vector field  : u = (u1, u2)");

// Deformation
//============================

mesh Th1 = movemesh(Th, [x+u1, y+u2]);
plot(Th , wait=1,  cmm="Dam before deformation");
plot(Th1, wait=1,  cmm="Dam after  deformation");

plot(Th, Th1, [u1,u2], wait=1,cmm="Deformed dam");

real u1max = u1[].max;
real u2max = u2[].max;
cout << "dep. max x = "<< u1max<< endl;
cout << "dep. max y = "<< u2max<< endl;
cout << "dep. au top = ( "<< u1(0,0)<<" , "<<u2(0,0)<<" )"<< endl;
```

Simplified model for a dam (5)

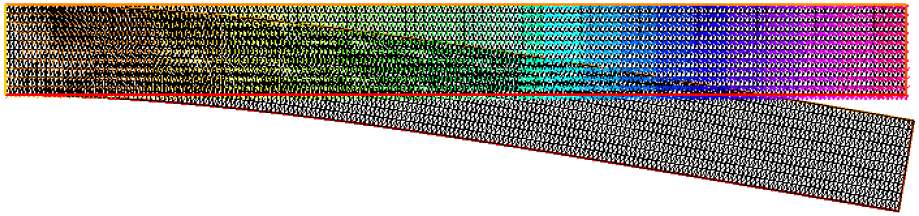
elast/dam_v02.edp

```
// Problem using macros
//============================

real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2] // EOM

macro div(u,v) ( dx(u)+dy(v) ) // EOM

problem pb([u1,u2],[v1,v2]) =
int2d(Th)(
  2.*mu*(epsilon(u1,u2)'*epsilon(v1,v2))
  +lambda*(div(u1,u2)*div(v1,v2))
         )
- int1d(Th, Gamma3)(p*v1)
+ on(Gamma1, u1=0, u2=0);
```

Basic features in FreeFem++

Deformation of a beam under its own weight

$\vec{f} = -g\vec{e}_y$

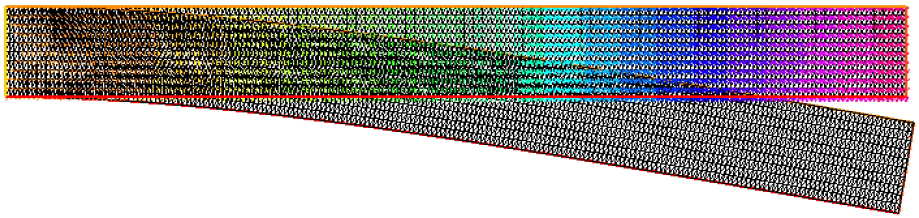


elast/beam_v01.edp

```
fespace Vh1(Th,P1);
fespace Vh(Th,[P1, P1]);
Vh [u1,u2], [v1,v2];
Vh1 p = -1;

// Problem using macros
//============================

real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2] // EOM
macro div(u,v) ( dx(u)+dy(v) ) // EOM
problem pb([u1,u2],[v1,v2]) =
int2d(Th)( 2.*mu*(epsilon(u1,u2)'*epsilon(v1,v2))
  +lambda*(div(u1,u2)*div(v1,v2)))
- int2d(Th)(p*v2)
+ on(4, u1=0, u2=0);
```

Basic features in FreeFem++

Model for the vibration of a beam



$$\frac{\partial^2 \vec{u}}{\partial t^2} - div(\vec{\sigma}) = \vec{f} \quad \text{in} \quad \Omega.$$

Initial condition $\vec{u}_0 = (0,0)$, $\left(\frac{\partial \vec{u}}{\partial t}\right)_0 = (0,-1)$

Weak formulation:

$$\int_\Omega \frac{\partial^2 \vec{u}}{\partial t^2}.\vec{v} + \int_\Omega \lambda (\nabla.\vec{u})(\nabla.\vec{v}) + \int_\Omega 2\mu\, \vec{\epsilon}(u) : \vec{\epsilon}(v) - \int_\Omega \vec{f}.\vec{v} - \int_{\partial\Omega} \left(\vec{\sigma}.\vec{n}\right).\vec{v} = 0,$$

Consider $\vec{f} = -g\vec{e}_y$ and $\left(\vec{\sigma}.\vec{n}\right)$ on the free borders.

We use an implicit scheme with 2nd order finite difference for:

$$\frac{\partial^2 \vec{u}}{\partial t^2} \approx \frac{\vec{u}^{n+1} - 2\vec{u}^n + \vec{u}^{n-1}}{(\delta t)^2}.$$

Vibration of a beam (1)

elast/beam_v02.edp

```
// Vectorial FE space
//===========================

fespace Vh1(Th,P1);

fespace Vh(Th,[P1, P1]);
Vh [u1,u2], [v1,v2];          // solution (n+1)
Vh [u1p,u2p], [u1pp,u2pp];// solutions (n) et (n-1)


// Time evolution
//===========================
real T=1000, dt=5, idt2=1/(dt*dt);
real [int] v0=[0,-1];    // initial velocity=du/dt
[u1pp, u2pp]=[0.,0.];    // cond initiale sur u
[u1p,u2p]=[u1pp,u2pp]+[dt*v0[0],dt*v0[1]]; // init condition using
    du/dt
```

Vibration of a beam (2)

elast/beam_v02.edp

```
// Time evolving problem (using macros)
//=====================================

real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2] // EOM

macro div(u,v) ( dx(u)+dy(v) ) // EOM

problem pbinst([u1,u2],[v1,v2]) =
int2d(Th)(
  2.*mu*(epsilon(u1,u2)'*epsilon(v1,v2))
  +lambda*(div(u1,u2)*div(v1,v2))
)
+int2d(Th)([u1,u2]'*[v1,v2]*idt2)
-int2d(Th)([u1p,u2p]'*[v1,v2]*2*idt2)
+int2d(Th)([u1pp,u2pp]'*[v1,v2]*idt2)
+ on(4, u1=0, u2=0);
```

Vibration of a beam (3)

elast/beam_v02.edp

```
// Time evolution
//==========================

for (real temps=2*dt;temps <=T; temps +=dt)
{
    Th1= movemesh(Th, [x+coef*u1, y+coef*u2]);
    plot(Th1,  cmm="time="+temps);
   cout << "time="<<temps<<" displacement at the top = ( "<< u1(0,L)<<"
        , "<<u2(0,L)<<" )"<< endl;

    pbinst;

   [u1pp,u2pp]=[u1p,u2p];
    [u1p,u2p]=[u1,u2];
}
```

See also the script elast/beam_v02.edp for a nicer visualisation!

Basic features in FreeFem++

Solving a non-linear simple PDE (1)

Consider the following non-linear problem:

$$-\nabla^2 u + \chi'(u) = f \quad \text{in } \Omega, \quad u = 0 \text{ on } \Gamma = \partial\Omega, \tag{10}$$

with χ' is the derivative of a given real function χ. With respect to the scalar product of $L^2(\Omega), \Omega \subset \mathbb{R}^2$,

$$< u, v >= \int_\Omega uv, \tag{11}$$

this problem corresponds to the minimization of the functional:

$$J(u) = \int_\Omega \frac{1}{2}\nabla(u).\nabla(u) + \chi(u) - fu. \tag{12}$$

The variational formulation of (10) is: find $u \in H_0^1(\Omega)$ such that

$$\forall v \in H_0^1(\Omega), \quad \int_\Omega \nabla(u).\nabla(v) + \chi'(u)v - \int_\Omega fv = 0 \tag{13}$$

Solving a non-linear simple PDE (2)

The Newton method to solve the non-linear equation $F(u) = 0$ consists in the iterative procedure:

$$u_{n+1} = u_n - \left[\frac{\partial F}{\partial u}(u_n)\right]^{-1} . [F(u_n)] . \tag{14}$$

Since u_n is known, solving the previous equation is equivalent to find $q_n = u_n - u_{n+1}$, solution of the PDE:

$$\left[\frac{\partial F}{\partial u}(u_n)\right] . q_n = F(u_n) \tag{15}$$

In our case:

$$F(u) = -\nabla^2 u + \chi'(u) - f,$$

and

$$\left[\frac{\partial F}{\partial u}(u_n)\right] . q = -\nabla^2 q + \chi''(u_n).q$$

Solving a non-linear simple PDE (3)

Since $q_n \in H_0^1(\Omega)$, the variational formulation of (15) becomes:

$$\forall v \in H_0^1(\Omega), \quad \int_\Omega \left[\nabla q_n.\nabla v + \chi''(u_n) q_n v\right] = \int_\Omega \left[\nabla u_n.\nabla v + \chi'(u_n) v - f v\right] \tag{16}$$

A good starting solution for the Newton method is the solution of the linear problem corresponding to (10).
We consider in the following

$$\chi(u) = u^4, \longrightarrow \chi'(u) = 4u^3, \quad \chi''(u) = 12u^2, \tag{17}$$

and, in order to have an exact solution

$$u_{ex}(x,y) = g(x,y) = g(x,y) = \sin(a_1 x) * \sin(a_2 y)$$

we set the second member

$$f(x,y) = (a_1^2 + a_2^2)\, g(x,y) + \chi'(g(x,y)). \tag{18}$$

Script for the non-linear equation (1)

nonlin/lap_nonlin_Newton.edp

```
// macros for chi
macro   chi(u) (pow(u,4))//
macro  dchi(u) (4*pow(u,3))//
macro ddchi(u) (12*pow(u,2))//

// exact solution
   Vh uex = sin(a1*x)*sin(a2*y);

// source term
   Vh fs = (a1*a1+a2*a2)*uex + dchi(uex);

// macro for energy
   macro energ(u,chis,fsource)
                  (int2d(Th)(0.5*(dx(u)*dx(u)+dy(u)*dy(u)))
                   +int2d(Th)(chis)-int2d(Th)(fsource*u))//
```

Script for the non-linear equation (2)

nonlin/lap_nonlin_Newton.edp

```
//=============================================================
// variational formulation of Newton method
//=============================================================

   Vh q;
   problem ANLIN(q,v) = int2d(Th)(graduv(q,v))
                      + int2d(Th)(ddchi(uold)*q*v)
                      - int2d(Th)(graduv(uold,v))
                      - int2d(Th)(dchi(uold)*v)
                      + int2d(Th)(fs*v)
                      + on(1,2,3,4,q=0);
```

See also the script `lap_nonlin_Newton_matr.edp` using a formulation with matrices!

Summary of Part II

Working with matrices

- **matrix**, **varf**,
- select the solver for the linear system (direct, iterative);
- up-to-date libraries (UMFPACK, MUMPS, PETSC), etc.

Summary of Part II

Working with matrices

- **matrix**, **varf**,
- select the solver for the linear system (direct, iterative);
- up-to-date libraries (UMFPACK, MUMPS, PETSC), etc.

Time evolving problems

- use you preferred finite-difference scheme;
- implicit/explicit schemes;
- Newton methods for fully implicit formulations.

Summary of Part II

Working with matrices

- **matrix**, **varf**,
- select the solver for the linear system (direct, iterative);
- up-to-date libraries (UMFPACK, MUMPS, PETSC), etc.

Time evolving problems

- use you preferred finite-difference scheme;
- implicit/explicit schemes;
- Newton methods for fully implicit formulations.

Vectorial FE spaces: fespace Vh(Th, [P1,P1])

- useful for many problems (elasticity, Stokes, etc)
- (warning) mixed entries in the arrays (following the dof);
- plot vector fields **plot([u1,u2]).**

Final remarks

Many other problems can be solved with FreeFem++

- incompressible fluid dynamics,
- Schroedinger (NLS) equation,
- eigenvalue problems,
- moving boundaries,
- domain decomposition methods (parallel computing),

... and many others!

Final remarks

Many other problems can be solved with FreeFem++

- incompressible fluid dynamics,
- Schroedinger (NLS) equation,
- eigenvalue problems,
- moving boundaries,
- domain decomposition methods (parallel computing),

... and many others!

Thank you for your attention, and ...

- use the doc pdf-file to find an example close to your application,
- check the web site and the mailing list,
- do not hesitate to contact us!