Advanced XLiFE++

Days 2016

june, 14-15, 2016

Unité de Mathématiques Appliquées
ENSTA ParisTech

Outline

- Vector unknown, multiple unknowns
- Understanding TermMatrix and TermVector
- Understanding Essential Conditions
- Edge element
- How to deal with time problem

Vector unknown

Vector unknown are involved when the space is a vector function spaces for instance

elastic displacement: $V = H^1(\Omega)^3$ **electric field :** $W = H(curl, \Omega)$

XLiFE++ does not deal with power of space but deal with Hrot (or Hdiv) :

```
Space W(omega, interpolation(_Nedelec,_firstFamily, 1, Hrot), "W", false);
Unknown E ( W, "E" );
```

any vector function $\mathbf{E}$ of space W has the following representation : $\mathbf{E} = \sum_i E_i \mathbf{w}_i$

scalar coefficient, vector basis

To deal with vector function of V, XLiFE++ use the following

```
Space V(omega, interpolation_Lagrange, 1, H1), "V", false);
Unknown u ( V, "u", 3 );   // 3-vector unknown
```

any vector function $\mathbf{u}$ of space V has the following representation : $\mathbf{u} = \sum_i \mathbf{u}_i \, w_i$

vector coefficient, scalar basis

Vector unknown

In both cases, unknowns are symbolically vector unknowns !

Differential operators, operations have to be consistent with vectors :

- **Implicit vector unknown e**

```
Space W(omega, interpolation(_Nedelec,_firstFamily, 1, Hrot), "W", false);
Unknown e( W, "E" ); TestFunction q( e, "q" );
BilinearForm a = intg(omega, curl(e)|curl(q)) - ome*intg(omega, e|q);   inner product
```

- **Explicit vector unknown u**

```
Space V(omega, interpolation_Lagrange, 1, H1), "V", false);
Unknown u ( V, "u", 3 );        // 3-vector unknown
TestFunction v( u, "v" );       // 3-vector test function
BilinearForm a = 2*mu*intg(omega, epsilon(u)%epsilon(v) )
                  + lambda*intg(omega, div(u)*div(v)) - omg2*intg(omega, u|v);
```

Access to a component of an explicit vector unknown

```
BilinearForm b = intg(omega, u(1)*v(1) );
```

vector unknown TermVector are stored as vector of vectors
vector unknown TermMatrix are stored as matrix of matrices

Multiple unknowns

It is easy to deal with problems with multiple unknowns of any kind

- **Set as many Unknowns as you want**
- **Deal with them in bilinear form as you want (staying consistent)**

$$a((u,\mathbf{p}),(v,\mathbf{q})) = \int_\Omega \mathbf{p}.\mathbf{q} + \int_\Omega u \,\mathrm{div}\,\mathbf{q} - \int_\Omega \mathrm{div}\,\mathbf{p}\, v$$

```
Space H(omega,interpolation(_Lagrange,_standard, k-1, H1),"H",true);
Space V(omega,interpolation(_RaviartThomas,_standard, k, Hdiv), "V", true);
Unknown p(V, "p");  TestFunction q(p, "q");          // p=grad(u)
Unknown u(H, "u");  TestFunction v(u, "v");
BilinearForm a=intg(omega, p|q) + intg(omega, u*div(q) ) - intg(omega, div(p)*v );
```

usage of vector unknowns, multiple unknowns have consequences on algebraic representation TermVector and TermMatrix

Understanding TermMatrix and TermVector

TermMatrix and TermVector are the core of XliFE++

- **TermMatrix mainly handles some matrix coefficients** $a(w_i, w_j)$
- **TermVector mainly handles some vector coefficients** $l(w_j)$

In order to deal with multiple unknowns in a transparent way, in fact

- **TermMatrix handles a list of SuTermMatrix, some** $a_{k,l}(w_{k,i}, w_{l,j})$
- **TermVector handles a list of SuTermVector, some** $l_k(w_{k,j})$

Su means Single unknown

Think about a block representation
a single unknown TermMatrix is 1x1 block

	u1	u2	u3
v1	M11	M12	
v2			
v3			

not necessarily squared

Understanding TermMatrix and TermVector

This block representation (list of SuTermMatrix) is called "local" representation

- **advantage : each SuTermMatrix has its own storage**
- **inconvenient : not well suited when factorize matrix or apply essential condition**

This is why in certain cicumstances , XLiFE++ is able to deal with a "global" representation mixing all the unknowns

it may be memory expansive !

As a consequence, the order of the unknowns may play a role.
They are ranked in the order of creation but you can change the order by setting the rank of an unknown :

```
Unknown p(V, "p");  TestFunction q(p, "q");
Unknown u(H, "u");  TestFunction v(u, "v");
setRanks(p, 4, u , 2);       // u is before p
```

TestFunction have same rank as it related unknown

Understanding TermMatrix and TermVector

- **Extract a block of a multiple unknown TermMatrix**

```
...
Unknown p(V, "p");  TestFunction q(p, "q");        // p=grad(u)
Unknown u(H, "u");  TestFunction v(u, "v");
BilinearForm a=intg(omega, p|q) + intg(omega, u*div(q) ) - intg(omega, div(p)*v );
TermMatrix A(a, "A");
TermMatrix Apq = A(p,q);
```

- **Setting storage of a multiple unknown TermMatrix, sets storage to all SuTermMatrix, may be to global representation**

- **Summing multiple unknowns TermMatrix may be complex process, summing and concatenating same unknowns SuTermMatrix and insert SuTermMatrix**

- **Product of a multiple unknowns TermMatrix by a multiple unknowns TermVector is done on each SuTermMatrix and produce a multiple unknowns TermVector having row unknowns of TermMatrix**

Similar management for TermVector

Essential conditions

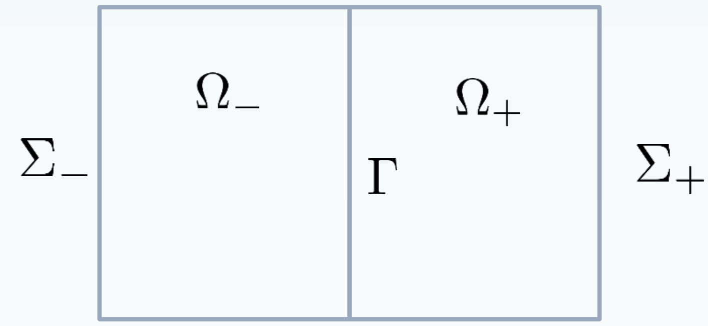
Essential conditions are **constraints in space**, main types :

Dirichlet condition : $u = 0 \text{ on } \Gamma$

Transmission condition : $[u] = g \text{ on } \Gamma$

Periodicity condition : $u_{|\Sigma+} = u_{|\Sigma-}$

Average condition : $\int_{\Omega_+} u = 0$



General expressions

lcop = val or fun on D **(one domain)**

lcop1 on D1 + lcop2 on D2 = val or fun **(two domains)**

lcop, lcop1, lcop2 are linear combination of operator on unknowns

D, D1, D2 are domains where act esssential conditions

```
Vector<Real> mapPM (const Point& P, Parameters& pa = defaultParameters)
{  Vector<Real> Q= P;    Q.y-=1;  return Q; }                map  Sigma+  ->  Sigma-
EssentialCondition  ecd= (u|sigmaM = 1) ;                         Dirichlet condition
EssentialCondition  ect =  (uM|gamma) - (uP|gamma) = f;  Transmission condition
defineMap(sigmaP, sigmaM, mapPM);
EssentialCondition  ecp = (u|sigmaP) - (u|sigmaM) = 0;       Periodic condition
```

Essential conditions

Collect **EssentialCondition** in a list : **EssentialConditions**

```
EssentialConditions ecs= (u|gammaM = 0) & (u|gammaP = 0)  & ((u|sigmaP) - (u|sigmaM) = 0);
EssentialConditions ecs= (uM|sigmaM = 1) & (uP|sigmaP = 1)  & ((uM|gamma) - (uP|gamma) = 0);
```

Conditions may be not consistent

General process to deal with many conditions (very powerful)

- Collect essential conditions in a constraints system : $C\,U = G$
- Reduce by QR algorithm to a minimal system : $U_E + D\,U_R = S$
- Reduce problem to solve $A\,U = F$ $\quad (A_R - A_E D)U_R = F - A_E S$

(for Dirichlet condition $u = 0 :\; D = 0 \longrightarrow A_R U_R = F$*)*

Detects and eliminates redundant or conflicting constraints

Average condition

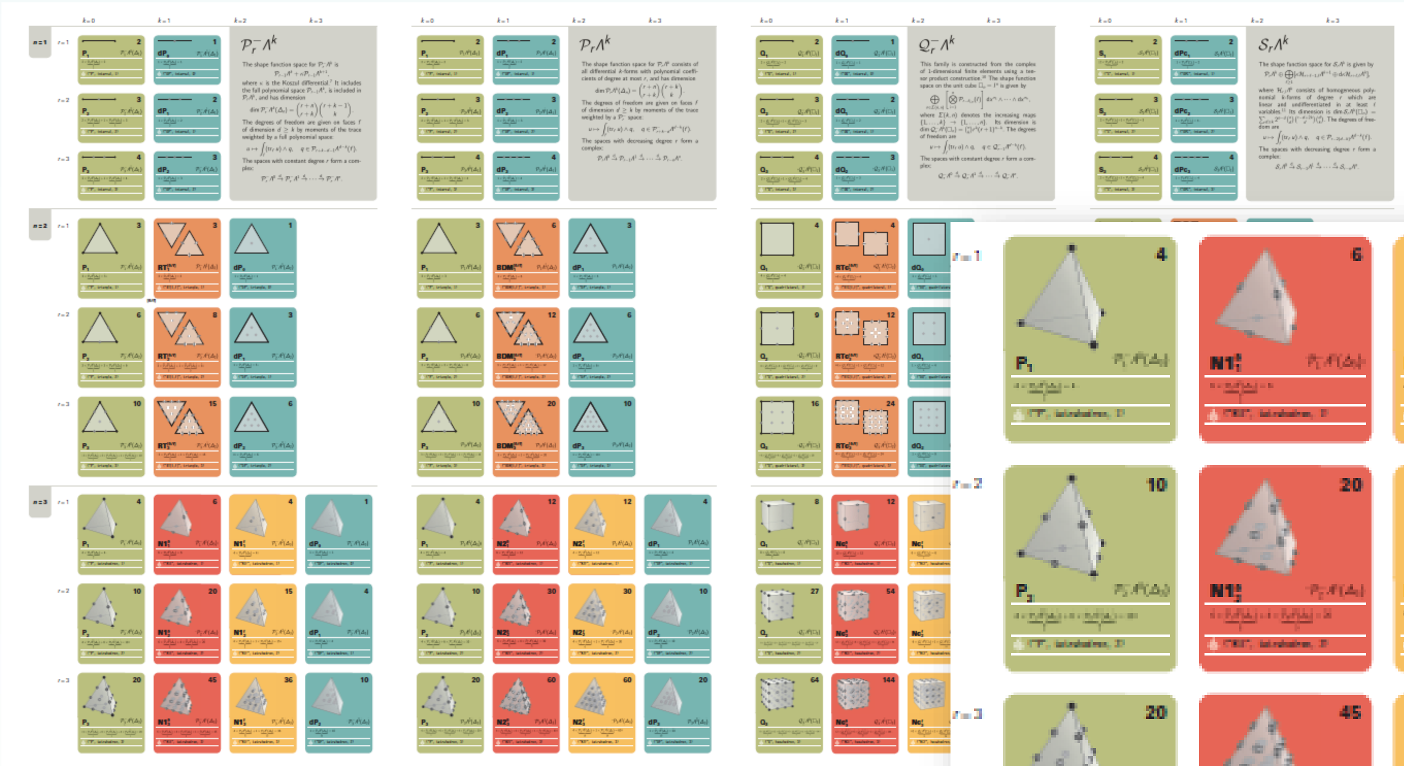
$$\int_\Sigma u = 0$$

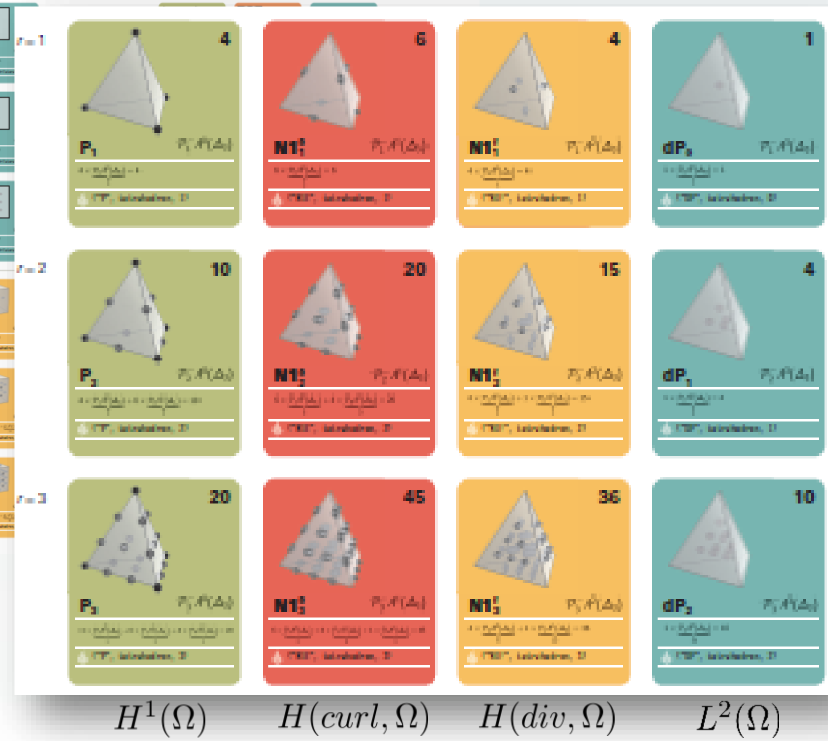
```
BilinearForm a=intg(omega, grad(u)|grad(v) );
TermMatrix A(a, intg(sigma,u) = 0 );
```

may be memory expansive and time expansive when global constraints

Edge elements

$H(curl,\Omega)$ and $H(div,\Omega)$ space approximations





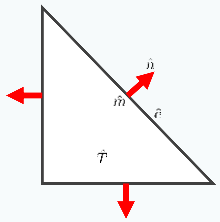
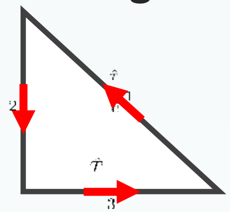
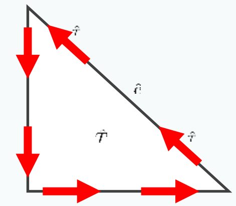
Nice periodic table of Finite Elements

Used in XLiFE++ to deal with any order Edge/Face elements

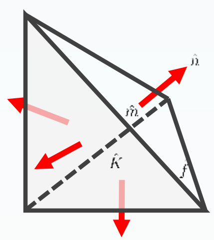
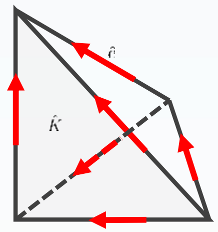
Edge/Face elements

Not all implemented !

- **on triangle : Raviart-Thomas, Nedelec Edge first and second family**

- **on tetrahedron : Nedelec Face and Nedelec Edge first families**

basis function are get from symbolic polynoms representation

Dofs are moment dofs but they have virtual coordinates

```
Space V (omega, interpolation(_Nedelec,_ firstFamily, 1, Hrot), "V", false);
Unknown e(V,"E"); TestFunction q(e,"q");
BilinearForm a = intg(omega, curl(e)|curl(q)) - ome*intg(omega, e|q);
EssentialConditions ecs = (ncross(e) | gamma)=0;
...
```

Time regim problem

There is no specific tool to deal with time regim problem!

BUT it is easy to deal with. Leap-frog scheme for wave equation

$$\begin{cases} U^{n+1} = 2U^n - U^{n-1} - M^{-1}\left((c\Delta t)^2 K U^n - (\Delta t)^2 F^n\right) & \forall n > 1 \\ U^0 = U^1 = 0 \text{ in } \Omega \end{cases}$$

```
TermMatrix A(intg(omega, grad(u)|grad(v) );
TermMatrix M(m=intg(omega, u * v) );
TermVector G(intg(omega, g*v) );                    // F = h(t)*G
// leap-frog scheme
Real c=1, dt=0.004, dt2=dt*dt, cdt2=c*c*dt2;
Number nbt=200; Real t=dt;
TermMatrix L;  ldltFactorize(M, L);
TermVector zeros(u, omega, 0.);
TermVectors U(nbt);
U(1)=zeros;  U(2)=zeros;
for(Number n=2; n<nbt; n++, t+=dt)
    U(n+1)=2*U(n)-U(n-1)-factSolve(L,cdt2*(A*U(n))-dt2*h(t)*G);
saveToFile("U",U,vtu);
```