



SORBONNE
UNIVERSITÉ

Inria

High-order Lagrange elements in FreeFem++

A. CHABIB, P-H. TOURNIER, F. HECHT

Laboratoire Jacques-Louis Lions
CNRS, Sorbonne Université UPMC , France
ALPINES team, Centre Inria de Paris

FreeFem Days
December 17, 2025

Finite element solution of time-harmonic wave propagation problems

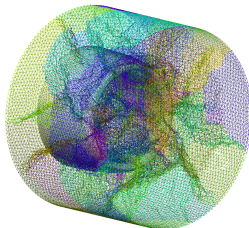
Helmholtz problem:

$$\begin{cases} -\Delta p - \omega^2 p = s & \text{in } \Omega \\ \text{Boundary conditions} & \text{on } \partial\Omega \end{cases}$$

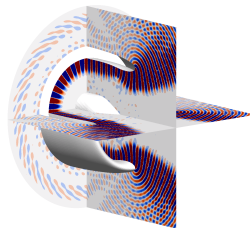
Finite element error:

$$\|p - p_h\|_{H^1} \leq C_1(\omega h)^k + C_2\omega^{(k+1)}h^k$$

Finite element mesh



Numerical field



High frequency (large ω)

Phenomena close to resonance

$\rightsquigarrow \begin{cases} \text{Fine mesh (small } h) \\ \text{High-order basis functions (large } k) \end{cases}$

Finite element solution of time-harmonic wave propagation problems

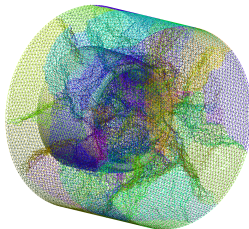
Helmholtz problem:

$$\begin{cases} -\Delta p - \omega^2 p = s & \text{in } \Omega \\ \text{Boundary conditions} & \text{on } \partial\Omega \end{cases}$$

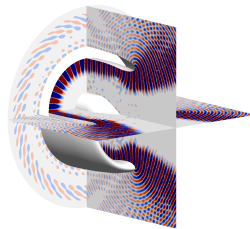
Finite element error:

$$\|p - p_h\|_{H^1} \leq C_1(\omega h)^k + C_2\omega^{(k+1)}h^k$$

Finite element mesh



Numerical field



High frequency (large ω)

Phenomena close to resonance






$\rightsquigarrow \begin{cases} \text{Fine mesh (small } h) & \checkmark \\ \text{High-order basis functions (large } k) & \text{X Limited to } P_4 \end{cases}$

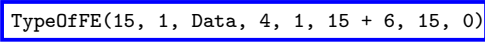
```
class TypeOfFE_P4Lagrange : public TypeOfFE {
public:
    static const int k = 4;
    static const int ndf = (k + 2) * (k + 1) / 2;
    static int Data[];
    static double Pi_h_coef[];
    static const int nn[15][4];
    static const int aa[15][4];
    static const int ff[15];
    static const int il[15];
    static const int jl[15];
    static const int kl[15];

    TypeOfFE_P4Lagrange( ) : TypeOfFE(15, 1, Data, 4, 1, 15 + 6, 15, 0)
    {
        static const R2 Pt[15] = {R2(0 / 4., 0 / 4.), R2(4 / 4., 0 / 4.),
                                   R2(0 / 4., 4 / 4.)...}

        // 3,4,5, 6,7,8, 9,10,11,
        int other[15] = {0, 1, 2, 5, 4, 3, 8, 7, 6, 11, 10, 9, 12, 13, 14};
        ...
    }
}
```

```

class TypeOfFE_P4Lagrange : public TypeOfFE {
public:
    static const int k = 4;  Interpolation order
    static const int ndf = (k + 2) * (k + 1) / 2;  # DOF =15
    static int Data[];  Topological information
    static double Pi_h_coef[];
    static const int nn[15][4];
    static const int aa[15][4];  Shape functions:  $\Phi_i = \prod_{j=0}^3 \frac{1}{ff[i]} (\lambda_{nn[i][j]} - aa[i][j])$  .hpp
    static const int ff[15];
    static const int il[15];
    static const int jl[15];  Nodes barycentric coordinates .hpp
    static const int kl[15];

    TypeOfFE_P4Lagrange( ) : TypeOfFE(15, 1, Data, 4, 1, 15 + 6, 15, 0)  Parent class constructor
    {
        static const R2 Pt[15] = {R2(0 / 4., 0 / 4.), R2(4 / 4., 0 / 4.),
                                   R2(0 / 4., 4 / 4.)...}

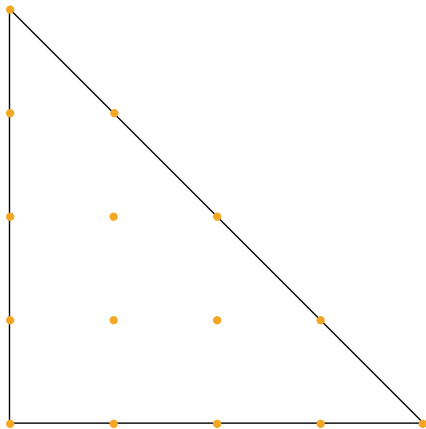
        // 3,4,5, 6,7,8, 9,10,11,
        int other[15] = {0, 1, 2, 5, 4, 3, 8, 7, 6, 11, 10, 9, 12, 13, 14};
        ...
    }
}

```

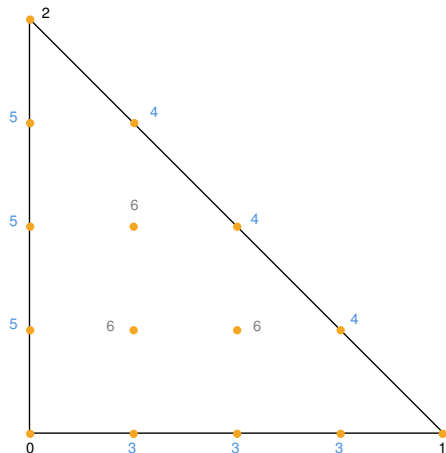
```
TypeOfFE_P4Lagrange( ) : TypeOfFE(15, 1, Data, 4, 1, 15 + 6, 15, 0)
```

Argument	Value P_4	Value P_k
# dof	15	$\frac{(k+1)(k+2)}{2}$
Dimension of the function space	1	1
Array containing topological information	Data	Data
Number of subdivisions for plotting	4	k
Number of component sub-finite elements	1	1
nombre de dof + dof permutation non sym sur aretes	15+6	cell9
Number of interpolation points	15	$\frac{(k+1)(k+2)}{2}$
dof dont on a la connaissance avant	0	0

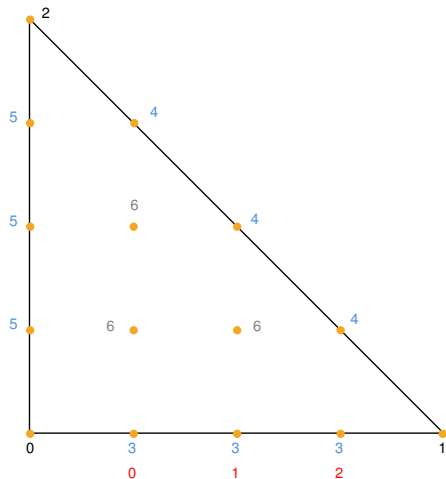
- 1: scalar function space(not vector field).
- Data: Numbering and relationship between DOFs and nodes.
- 4: To visualize the element, it's subdivided into k sub-elements.
- The 15 points where functions are evaluated to construct the interpolant.



```
int TypeOfFE_P4Lagrange::Data[] = {  
    // the support number of the node of the dof  
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,  
    // the number of the dof on the support  
    0, 0, 0, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2,  
    // the node of the dof  
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,  
    // the dof come from which FE (generally 0)  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    // which are de dof on sub FE  
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,  
    // for each compontant $j=0,N-1$ it give the sub FE ass  
    0,  
    // First dof  
    0,  
    // #dof  
    15  
};
```

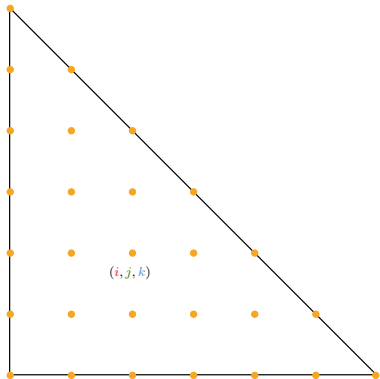


```
int TypeOfFE_P4Lagrange::Data[] = {
    // the support number of the node of the dof
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
    // the number of the dof on the support
    0, 0, 0, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2,
    // the node of the dof
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
    // the dof come from which FE (generally 0)
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    // which are de dof on sub FE
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    // for each compontant $j=0,N-1$ it give the sub FE ass
    0,
    // First dof
    0,
    // #dof
    15
};
```

```
int TypeOfFE_P4Lagrange::Data[] = {
    // the support number of the node of the dof
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
    // the number of the dof on the support
    0, 0, 0, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2,
    // the node of the dof
    0, 1, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
    // the dof come from which FE (generaly 0)
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    // which are de dof on sub FE
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    // for each compontant $j=0,N-1$ it give the sub FE ass
    0,
    // First dof
    0,
    // #dof
    15
};
```

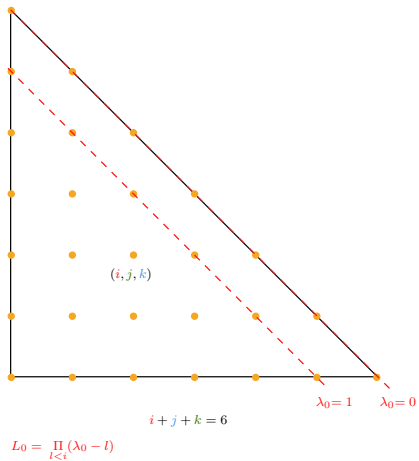
Shape functions



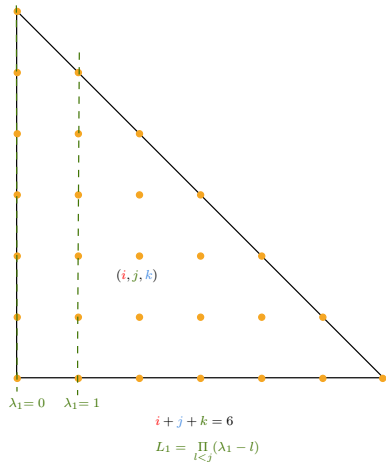
$$i + j + k = 6$$

Let i , j , and k be the barycentric coordinates of a given node.

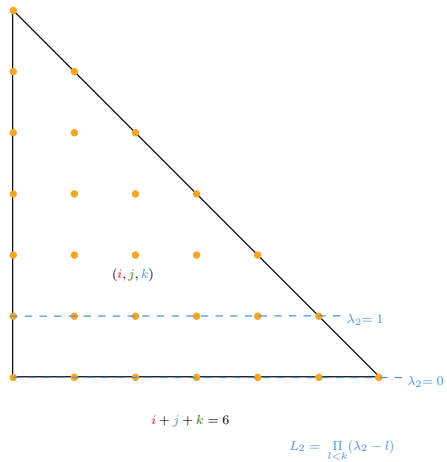
Shape functions



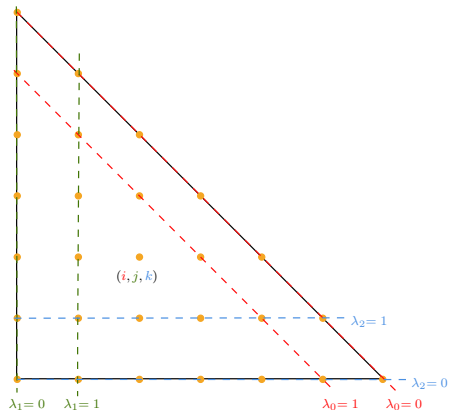
Shape functions



Shape functions



Shape functions



$$i + j + k = 6$$

$$L_0 = \prod_{l < i} (\lambda_0 - l) \quad L_1 = \prod_{l < j} (\lambda_1 - l) \quad L_2 = \prod_{l < k} (\lambda_2 - l)$$

$$L = \frac{1}{i!j!k!} L_0 \times L_1 \times L_2$$

```
void BasisFctPK(int , vector<vector<long>> &nn,
               vector<vector<long>> &aa,
               vector<long> &ff) {
    int idx = 0;
    for (auto &coordinate : coordinate_list) {
        int i = coordinate[0];
        int j = coordinate[1];
        int k = coordinate[2];
        if (i + j + k == p) {
            int ID = 0;
            ff[idx] = factorial(i)*factorial(j)
                    *factorial(k);
            if (i > 0) {
                for (int ii = 0; ii < i ; ii++) {
                    nn[idx][ID] = 0;
                    aa[idx][ID] = ii;
                    ID++;
                }
                // same for j and k
                idx++;
            }
        }
    }
}
```

- ☐ Declaration of a subclass `TypeOfFE_PkLagrange` of `TypeOfFE`
- ☐ Dynamic allocation of arrays depending on the interpolation order
- ☐ Arrays are filled on the fly during the creation of the object.

- Declaration of a subclass `TypeOfFE_PkLagrange` of `TypeOfFE`
- Dynamic allocation of arrays depending on the interpolation order
- Arrays are filled on the fly during the creation of the object.

- **Once the attributes are filled, the elements can be added to the DSL using:**

```
static TypeOfFE_PkLagrange PKLagrange(k); // Obj of the class TypeOfFE_PkLagrange
namespace Fem2D {
    ...
    static void init() {

        AddNewFE("PKLagrange", &PKLagrange);
        static ListOfTFE FE_PK("PKLagrange", &PKLagrange); //add Pk to the list of Common FE
    }
} // namespace Fem2D
LOADFUNC(Fem2D::init);
```


- Declaration of a subclass `TypeOfFE_PkLagrange` of `TypeOfFE`
- Dynamic allocation of arrays depending on the interpolation order
- Arrays are filled on the fly during the creation of the object.

- **Once the attributes are filled, the elements can be added to the DSL using:**

```
static TypeOfFE_PkLagrange PKLagrange(k); // Obj of the class TypeOfFE_PkLagrange
namespace Fem2D {
    ...
    static void init() {

        AddNewFE("PKLagrange", &PKLagrange);
        static ListOfTFE FE_PK("PKLagrange", &PKLagrange); //add Pk to the list of Common FE
    }
} // namespace Fem2D
LOADFUNC(Fem2D::init);
```

Integrating polynomials of arbitrary order?

Grundmann-Möller quadrature formula

Let $s \in \mathbb{N}$, $d = 2s + 1$, and $n \in \mathbb{N}$ a given dimension. Then:

$$I_n(f) = \sum_{i=0}^s (-1)^i 2^{-2s} \frac{(d+n-2i)^d}{i!(d+n-i)!} \sum_{\substack{|\beta|=s-i \\ \beta_0 \geq \dots \geq \beta_n}} \sum f\left(\frac{2\beta_1+1}{d+n-2i}, \dots, \frac{2\beta_n+1}{d+n-2i}\right)$$

is an invariant integration formula of degree d for the function f on an n -simplex,
where: $\beta = (\beta_0, \beta_1, \dots, \beta_n)$, and $|\beta| = \beta_0 + \beta_1 + \dots + \beta_n$

Grundmann-Möller quadrature formula

Let $s \in \mathbb{N}$, $d = 2s + 1$, and $n \in \mathbb{N}$ a given dimension. Then:

$$I_n(f) = \sum_{i=0}^s (-1)^i 2^{-2s} \frac{(d+n-2i)^d}{i!(d+n-i)!} \sum_{\substack{|\beta|=s-i \\ \beta_0 \geq \dots \geq \beta_n}} \sum f\left(\frac{2\beta_1+1}{d+n-2i}, \dots, \frac{2\beta_n+1}{d+n-2i}\right)$$

is an invariant integration formula of degree d for the function f on an n -simplex,
where: $\beta = (\beta_0, \beta_1, \dots, \beta_n)$, and $|\beta| = \beta_0 + \beta_1 + \dots + \beta_n$

GM formula:

- ☐ is capable of handling any arbitrary polynomial order
- ☐ is exact for polynomials of degree d
- ☐ is easily adaptable to any spatial dimension n
- ☐ does not guarantee that all weights are positive
- ☐ is prone to computational overflow issues
- ☐ is prone to numerical stability issues



Conclusion:

- ☐ Analysis of the existing 2D Lagrange finite elements
- ☐ Design of a generic 2D P_k element inspired by existing elements
- ☐ Development of associated quadrature formulas

Outlooks:

- ☐ More rigorous verification of convergence
- ☐ Addressing numerical instability in quadrature formulas
- ☐ Extension to the case of 3D Lagrange