High-Order Finite Elements in Diffpack

Xing Cai Klas Samuelsson

September 29, 2003

Abstract

This short note gives a brief introduction to the current implementation of the p-version of the finite element method in Diffpack. We explain the construction of the non-standard triangular and tetrahedral high-order elements, which distribute the nodes uniformly on the element edges, sides, and in the interior. Examples are provided to demonstrate how to use the corresponding p-version of the finite element method in Diffpack.

1 Introduction

The standard h-version of the finite element method (FEM) has been widely used for solving partial differential equations (PDEs). That is, basis functions of the first or second order are used in the discretization, where decreasing the element size h is the main strategy for obtaining an improved accuracy of the solution. The h-version has the advantage of giving rise to linear systems with only a small number of nonzero entries per row in the system matrix, therefore very efficient for iterative methods. However, reducing h is normally done through grid refinement, which may be a complicated procedure for many solution domains. Moreover, to obtain sufficient accuracy by the h-version, it is often necessary to refine a starting grid many times, giving rise to a large number of elements and thus a system that consists of a large number of linear equations.

On the other hand, the so-called p-version of the FEM is based on the idea of employing basis functions of higher orders when better accuracy is desired. The p-version is particularly advantageous for treating certain problems, such that exponential rates of convergence can be obtained. Arising from the combination of the h- and p-version, the so-called hp-FEM is a more robust method, which controls both the local element size h and the order of the basis functions.

Ever since the beginning, Diffpack [2] has contained a finite element library for the standard h-FEM. A toolbox for adaptivity was added several years ago to Diffpack's finite element library, so that adaptive grid refinement in both 2D and 3D is available for the h-FEM. Recently, an add-on p-FEM module has been programmed on the basis of the h-FEM library and the adaptivity toolbox. It is the purpose of this note to provide the readers with a brief overview of the

underlying numerics and the usage of the p-FEM module. The topic of hp-adaptivity will be covered in [1]

2 High-Order Basis Functions

Definition of high-order basis functions is the starting point for the p-FEM. We will only consider 2D triangular and 3D tetrahedral elements. We use a non-standard definition of the high-order basis functions, where the nodes are uniformly distributed inside each element. To illustrate this feature, let us look at Figure 1. We can see that the element contains 5th-order basis functions and has in total 21 nodes, where 3 nodes are the standard element vertex nodes, 12 nodes are the so-called *inner-edge* nodes lying on the three element edges, and the remaining 6 nodes are so-called *interior* nodes lying entirely in the interior of the triangular element.

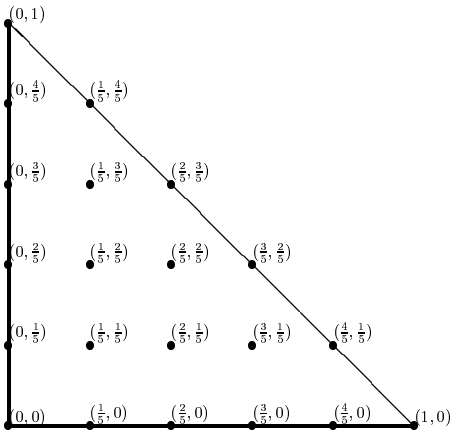

Figure 1: An example of a triangular element using uniformly distributed nodes.

2.1 The Mathematical Formulation

We explain in the following how the high-order basis functions are mathematically defined for the 2D triangular elements. The 3D high-order basis functions are defined in the same fashion.

For simplicity, let us consider a triangle with vertices at $(1,0)$, $(0,1)$, and $(0,0)$. (The same mathematical principle applies to any triangles.) We denote by $N_i(x,y)$ the ith basis function, associated with the ith node in the element. In case of a linear element, i.e., with the three vertices as the nodes, the three basis functions are

$$\begin{aligned} N_1(x,y) &= x, \\ N_2(x,y) &= y, \end{aligned}$$

$$N_3(x,y) \quad = \quad 1-x-y.$$

Here, it is convenient to introduce a third coordinate $w = 1-x-y$, so that $N_3 = w$. We observe that N_1 is a first-degree polynomial in x, N_2 is a first-degree polynomial in y, and N_3 is a first-degree polynomial in w.

The way of building high-order basis functions is to introduce new nodes into the linear element, and we distribute the new nodes uniformly. That is, for an element containing pth-order basis functions, we place $p-1$ new nodes uniformly on each edge between the two vertices, and the new interior nodes are placed accordingly, see Figure 1. The total number of nodes in such a high-order triangular element is

$$n_p^{\text{total}} \quad = \quad \frac{p(p+1)}{2}.$$

In this setup, we want to define for each node a basis function that is a polynomial of degree p. The high-order basis functions should also satisfy the following requirement:

$$\begin{aligned} N_i(x_i,y_i) &= 1 \quad \text{where } (x_i,y_i) \text{ is } i\text{th node's location}, \\ N_i(x_j,y_j) &= 0 \quad j \neq i, \\ \sum_{i=1}^{n_p^{\text{total}}} N_i(x,y) &= 1 \quad \forall (x,y) \in \Omega_e. \end{aligned} \tag{1}$$

In general, such high-order basis functions of order p are of the form:

$$N_i(x,y,w) \quad = \quad C_i\, f_x^{p_x}(x) f_y^{p_y}(y) f_w^{p_w}(w), \tag{2}$$

where $f_x^{p_x}(x)$, e.g., denotes a polynomial of degree p_x in x. That is, $N_i(x,y,w)$ is a product of three polynomials. Clearly, we require

$$p_x \geq 0,\ p_y \geq 0,\ p_w \geq 0, \text{ and } p_x + p_y + p_w = p.$$

The polynomials $f_x^{p_x}(x)$, $f_y^{p_y}(y)$, and $f_w^{p_w}(w)$ all have coefficient 1 for the leading term, and the coefficient C_i is chosen such that N_i attains value 1 on the ith node.

An important observation is that each node in a triangular pth-order element, except the three vertices, sits on the cross point of three lines:

$$x = \frac{k_1}{p}, \quad y = \frac{k_2}{p}, \quad \text{and} \quad w = \frac{k_3}{p}, \tag{3}$$

where $0 \leq k_1, k_2 \leq p-1$, and $k_3 = p - k_1 - k_2$, see Figure 2. In addition, several nodes align on each of the lines in (3). Consequently, if $y = k_2/p$ is chosen as a zero in the polynomial $f_y^{p_y}(y)$, then $f_y^{p_y}(y)$ attains value 0 on all the $p+1-k_2$ nodes that lie on the line $y = k_2/p$.

Defining a basis function N_i that is of form (2) and satisfies (1) is the same as finding $f_x^{p_x}(x)$, $f_y^{p_y}(y)$, and $f_w^{p_w}(w)$, which have zeros on p lines of form (3).

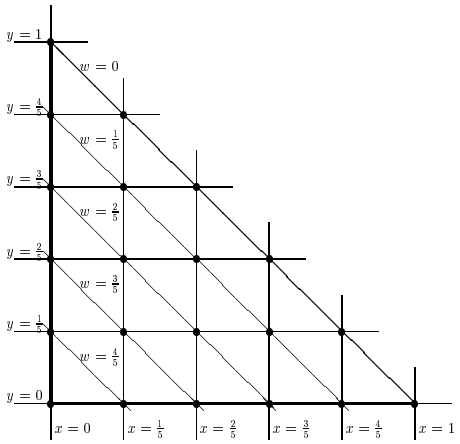

Figure 2: Each node in a triangular pth-order element, except the three vertices, sits on the cross point of three lines: $x = k_1/p$, $y = k_2/p$, and $w = k_3/p$.

The result is such that N_i attains value 0 on all the nodes except node number i. For example, when we want N_i to attain value 0 on all the nodes that are aligned on $x = 3/p$, we can achieve this by choosing $x = 3/p$ as the zero of the polynomial $f_x^{p_x}(x)$, i.e., $f_x^{p_x}(x) = (x - \frac{3}{p}) f_x^{p_x - 1}(x)$.

As a concrete example, let us consider the fifth-order case shown in Figure 1, where each edge has thus 4 nodes between the two vertices, and there are 6 interior nodes, all of them are uniformly placed. Suppose we want to build the basis function that is associated with node $(1, 0)$. This can be achieved by using

$$N(x, y, w) \quad = \quad \frac{625}{24} \cdot x \cdot \left(x - \frac{1}{5}\right) \cdot \left(x - \frac{2}{5}\right) \cdot \left(x - \frac{3}{5}\right) \cdot \left(x - \frac{4}{5}\right).$$

It can be verified that the above basis function attains value 1 at $(1, 0)$, whereas it attains value 0 at all the other nodes. Similarly, for the basis function associated with node $(\frac{1}{5}, \frac{2}{5})$, it can be found as

$$N(x, y, w) \quad = \quad \frac{3125}{4} \cdot x \cdot y \cdot \left(y - \frac{1}{5}\right) \cdot w \cdot \left(w - \frac{1}{5}\right).$$

2.2 Summary of Available High-Order Elements

In Table 1, we give an overview of ten triangular elements with an increasing order of basis functions, from $p = 1$ to $p = 10$. We denote by n_p^{total} the total number of nodes in an element of order p, whereas n_p^{edge} denotes the number of inner-edge nodes, and n_p^{interior} denotes the number of interior nodes.

The construction of tetrahedral elements using high-order basis functions is similar to that in the 2D case. We also distribute the nodes uniformly inside each tetrahedron. That is, the distribution of nodes on each of the four sides of

p	n_p^{total}	n_p^{vertex}	n_p^{edge}	n_p^{interior}
1	3	3	0	0
2	6	3	3	0
3	10	3	6	1
4	15	3	9	3
5	21	3	12	6
6	28	3	15	10
7	36	3	18	15
8	45	3	21	21
9	55	3	24	28
10	66	3	27	36

Table 1: Overview of triangular elements of different orders; n_p^{total} denotes the total number of nodes in each element, n_p^{vertex} denotes the number of vertex nodes, n_p^{edge} denotes the number of inner-edge nodes, and n_p^{interior} denotes the number of interior nodes.

a tetrahedral element is the same as that in a triangular element. The nodes lying entirely in the interior volume are placed accordingly. In Table 2, we summarize ten tetrahedral elements using basis functions of an increasing order, from $p = 1$ to $p = 10$. Note that the so-called *inner-side* nodes of a tetrahedral element are the same as the interior nodes in a triangular element, whereas the interior nodes of a tetrahedral element lie entirely inside the 3D volume of the element. We remark that a possible numbering scheme of all the basis functions in each element can start with the vertex nodes, continue with the inner-edge nodes, followed by the inner-side nodes, and end with the interior nodes. The numbering process typically adopts a "circle-in" strategy.

p	n_p^{total}	n_p^{vertex}	n_p^{edge}	n_p^{side}	n_p^{interior}
1	4	4	0	0	0
2	10	4	6	0	0
3	20	4	12	4	0
4	35	4	18	12	1
5	56	4	24	24	4
6	84	4	30	40	10
7	120	4	36	60	20
8	165	4	42	84	35
9	220	4	48	112	56
10	286	4	54	144	84

Table 2: Overview of tetrahedral elements of different orders; n_p^{total} denotes the total number of nodes in each element, n_p^{vertex} denotes the number of vertex nodes, n_p^{edge} denotes the number of inner-edge nodes, n_p^{side} denotes the number of inner-side nodes, and n_p^{interior} denotes the number of interior nodes.

3 New Functionality Associated with p-FEM

3.1 Summary of New Classes

The following classes are developed for incorporating the p-FEM into Diffpack. All the new classes, except `GridUtil`, are invisible to the users.

- `GridUtil`: a utility repository class containing mainly different versions of the member function `makeHigherOrderGrid`, which can be used to create a high-order finite element grid based on a linear grid.
- `ElmT10n2D`: triangular element with $p = 3$, as a subclass of `ElmDef`.
- `ElmT15n2D`: triangular element with $p = 4$, as a subclass of `ElmDef`.
- `ElmT21n2D`: triangular element with $p = 5$, as a subclass of `ElmDef`.
- `ElmT28n2D`: triangular element with $p = 6$, as a subclass of `ElmDef`.
- `ElmT36n2D`: triangular element with $p = 7$, as a subclass of `ElmDef`.
- `ElmT45n2D`: triangular element with $p = 8$, as a subclass of `ElmDef`.
- `ElmT55n2D`: triangular element with $p = 9$, as a subclass of `ElmDef`.
- `ElmT66n2D`: triangular element with $p = 10$, as a subclass of `ElmDef`.
- `ElmT20n3D`: tetrahedral element with $p = 3$, as a subclass of `ElmDef`
- `ElmT35n3D`: tetrahedral element with $p = 4$, as a subclass of `ElmDef`
- `ElmT56n3D`: tetrahedral element with $p = 5$, as a subclass of `ElmDef`
- `ElmT84n3D`: tetrahedral element with $p = 6$, as a subclass of `ElmDef`
- `ElmT120n3D`: tetrahedral element with $p = 7$, as a subclass of `ElmDef`
- `ElmT165n3D`: tetrahedral element with $p = 8$, as a subclass of `ElmDef`
- `ElmT220n3D`: tetrahedral element with $p = 9$, as a subclass of `ElmDef`
- `ElmT286n3D`: tetrahedral element with $p = 10$, as a subclass of `ElmDef`
- `ElmDef2_prm`: new "container" for all the parameters needed for creating one object of the above subclasses of `ElmDef`.
- `QuadratureRules`: a utility class containing special quadrature rules associated with the high-order elements. We remark that quadrature rules using many evaluation points have to be used to keep up the accuracy of a numerical integration with that of a high-order element.

3.2 Creating a High-Order Finite Element Grid

For a Diffpack user who wants to adopt the p-FEM for solving a PDE, the only major modification that she has to introduce to a standard Diffpack simulator is the creation of a finite element grid having elements with desired order p. The approach is as follows:

- First, create a standard finite element grid containing linear triangular or tetrahedral elements. Note that the constructed grid object must be of type `GridFEAdT`, which is available from the adaptivity toolbox. For example,

```
Handle(GridFEAdT) linear_grid = new GridFEAdT;
String gridfile = menu.get ("gridfile");
readOrMakeGrid (*linear_grid, gridfile);
```

  We note that `linear_grid` can undergo (adaptive) h-refinements if necessary, before being used as a starting grid for creating a high-order grid.

- Then, the `makeHigherOrderGrid` member function of class `GridUtil` should be applied to create a high-order finite element grid, e.g.,

```
GridFE* high_order_grid = new GridFE;
const int p_order = 3;
GridUtil::makeHigherOrderGrid (high_order_grid, *linear_grid, p_order);
```

  The result is that `high_order_grid` is created to contain elements of order $p = 3$.

- Thereafter, the new high-order grid `high_order_grid` should be used for all the subsequent computations, which can retain the original implementation. The only exception is the storage of a solution field using Diffpack's `SaveSimRes` class. This is because high-order finite element grid can not be handled by any standard visualization tools. A remedy is to "map" a high-order finite element grid back to a linear grid using exactly the same nodes. This mapping operation can be carried out by the `makeLinearGridUsingSameNodes` function in class `GridUtil`. So a code segment for achieving this can be as follows:

```
Handle(GridFEAdT) linear_grid = new GridFEAdT;
GridUtil::makeLinearGridUsingSameNodes (linear_grid.getPtr(),
                                        *high_order_grid);
Handle(FieldFE) linear_field = new FieldFE (*linear_grid,
                                            "linear field");
Handle(DegFreeFE) linear_dof = new DegFreeFE (*linear_grid, 1);
linear_dof->vec2field (solution_vec, *linear_field);
database->dump (*linear_field);  // database is an object of SaveSimRes
```

References

[1] X. Cai and K. Samuelsson. Using *hp*-adaptivity in Diffpack. Technical report, Simula Research Laboratory, 2004.

[2] H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Textbooks in Computational Science and Engineering. Springer, 2nd edition, 2003.