International Journal of Trend in Scientific Research and Development (IJTSRD)
International Open Access Journal | www.ijtsrd.com
ISSN No: 2456 - 6470 | Volume - 2 | Issue – 6 | Sep – Oct 2018

# In Search for the Super Element: Algorithms to Generate Higher Order Elements

**Mohammad Tawfik**
Academic Director, Academy of Knowledge, Egypt

**ABSTRACT**
This work was motivated by the need that may arise during the creation of finite element programs, for higher order elements. The problem of selecting or creating shape functions that satisfy the required need of the problem may be one major problem that stand in the way of the element creation. In this work we present an attempt that points in the direction of creating the element matrices for higher order elements using simple, Lagrange, and modified Lagrange polynomials. The results obtained for the test cases indicate the possibilities and limitations on those attempts. It is concluded that the modifies Lagrange polynomials have a great potential for use in elements with high number of nodes and/or high number of DOF per node. Nevertheless, they impose a high computational cost on the element matrices generation.

***Keywords:*** *higher order elements, finite element analysis*

## 1. INTRODUCTION

In the past decades, the finite element method has grown into a well developed numerical technique that can be used to provide accurate and relatively quick solution to boundary value problems that describe several physical problems. The procedure of the finite element model starts by dividing the domain into *elements* that are connected through *nodes*. Each node has *degrees of freedom* (DOF) that, usually, present the value and derivatives of the physical function of interest. Then the elements are defined by the *interpolation* functions that describe the change of the function within their boundaries. Following that, the whole domain description is created by assembling the elements and applying the boundary conditions. Finally, the problem is solved to obtain the values of the DOF of all the nodes. 1, 2

The basic assumption is that as the number of elements increase in the domain, the solution approaches the exact one. The power of the finite element method lies in that it may use simple functions, usually polynomials, to describe the change of the target function inside an element, and then use that to describe the change in the whole domain by *assembling* the different elements that cover the domain.

Another approach to reach higher accuracy is to increase the order of each of the elements used in the model which reduces the need for higher number of elements (p-version finite element). Higher order elements may be generated by using more nodes in the element, using more complex functions, or by *hierarchical* elements. 3, 4, 5, 6

Most of the work performed for the solution of the finite element problem is algorithmic, that enabled the excessive use computers and the production of computer packages that can manipulate several physical problems with complex domain geometries. Nevertheless, in the core of all finite element codes, lies the element matrices. The element matrices are generated through the knowledge of the mathematical model that describes the physical problem and the selection of the interpolation functions, *shape functions*. Equipped with those two pieces of information, you may generate the element matrices for any physical problem. However, the generation of a general element is still far from algorithmic. 7

A special problem faces the generation of the finite element model is the selection of the interpolation function that satisfies the prescribed degrees of freedom at the nodes 5, 6, 8. Many of the problems solved involved the *selection* of functions that are published in the literature most of which are the so called Lagrange Polynomials. The Lagrange polynomials possess a preferred characteristic that they have the value of one at a specific point, node, in the domain and the value of zero at all other nodes. When the DOF involve derivatives of the function, the Lagrange polynomials should be selected to satisfy the condition that they have the slope of one at the associated node and zero at the others as well. However, it is quite difficult to find the Lagrange polynomials that may be used in a general problem that is why they may be generated by using classical interpolation techniques.

The classical interpolation techniques require the evaluation of the polynomial coefficients by *forcing* the polynomial to satisfy the boundary conditions, thus, requiring the solution of a set algebraic equation, hence, the inversion of a matrix. The matrix inversion process is a straight forward numerical technique; however, as the number of variables increase, the matrix eventually becomes singular because of the round-off errors. 9, 10

Another challenge that faces the creators of finite element models is the performance of integration. The element matrices are generated by integrating the shape functions or their derivatives over the element. In many simple cases, the integration is readily available by hand or using symbolic manipulators. However, in more practical cases, numerical integration is required which, in turn, introduces errors to the resulting element models.

In the past years, several research articles were published, mostly by mathematics oriented researchers, about the topic of generating higher order polynomials, shape functions, for applications in the finite element models 5, 6, 8, 11, 12. Such research, among many others not cited here, accomplished a great task in the direction of generalizing the automation of finite element model generation. Following that, an excellent attempt for the automation was presented in 7 using the symbolic manipulator Mathematica® and an accompanying package AceGen®.

None of the research reviewed by the author presented a straight-forward methodology that may be used by the engineers who need to create finite element models on numerical manipulators, thus, in the following work, we will be attempting to describe a generalize procedure for the generation of the finite element matrices with an eye on elements with large numbers of nodes and degrees of freedom. We will start by describing a classical method of generating the matrices and creating the shape functions, then we will move towards describing attempts to generate a general method for performing exact integration for generating element matrices. Following that, we will describe the attempts using Lagrange polynomials to avoid matrix inversion, finally, modifies Lagrange polynomials will be created to enable to the creation of different finite element problems.

The main purpose of this work is providing a methodology that may by applied using numerical manipulators, especially open source, to generate the element matrices that may be used by researchers without the need for commercial packages or commercial symbolic manipulators.

## 2. CLASSICAL APPROACH

### 2.1. Interpolation Polynomials – Shape Functions

The finite element model starts by assuming the solution of the unknown function in terms of an interpolation polynomial that satisfies the values of the degrees of freedom at the nodes of the element. In most text, these functions are denoted the letter *N* such that:

$$f(x) = f_1 N_1(x) + f_2 N_2(x) + \dots$$

Where $f_i$ are the values of the function at node *i* and $N_i(x)$ are polynomials that have the value of one at the corresponding node and zero at every other node. The above relation may be written as:

$$f(x) = \lfloor N_1(x) \quad N_2(x) \quad \dots \rfloor \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \end{Bmatrix} = \lfloor N(x) \rfloor \{\delta\}$$

The first problem that faces the researcher who wants to create a new finite element model is to find and select the polynomials that may be used for the problem. However, that is not a great problem for low order elements since the polynomials are readily available in many texts. Nevertheless, coding them may involve mistyping which will require debugging of the code. On the other hand, it becomes a bit inconvenient to change the number of nodes per element to test the effect of higher, or lower, order elements. Thus we may use another way of defining the polynomials that can reduce both problems, that is by using regular simple polynomials such that:

$$f(x) = a_0 + a_1 x + a_2 x^2 \ldots = \lfloor 1 \quad x \quad x^2 \quad \ldots \rfloor \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \end{Bmatrix} = \lfloor H(x) \rfloor \{a\}$$

Where $a_i$ are constants, generalized coordinates that should be determined for the function to satisfy the given values. To do that we will need to set a number of equations and solve them to find those values, thus:

$$f(x_1) = f_1 = \lfloor H(x_1) \rfloor \{a\}$$
$$f(x_2) = f_2 = \lfloor H(x_2) \rfloor \{a\}$$
$$f(x_3) = f_3 = \lfloor H(x_3) \rfloor \{a\}$$
$$\vdots$$

Which may be written as:

$$\begin{bmatrix} \lfloor H(x_1) \rfloor \\ \lfloor H(x_2) \rfloor \\ \lfloor H(x_3) \rfloor \\ \vdots \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{Bmatrix}$$
$$[T]\{a\} = \{\delta\}$$

Which gives:

$$\{a\} = [T^{-1}]\{\delta\}$$
$$f(x) = \lfloor H(x) \rfloor [T^{-1}]\{\delta\}$$

It can be readily proven that:

$$\lfloor N(x) \rfloor = \lfloor H(x) \rfloor [T^{-1}]$$

Thus using this procedure we were able to obtain the polynomials for any number of degrees of freedom without having to look them up in literature or enduring the problems of mistyping them into the code. However, nothing comes for free; using this method will involve matrix inversion which *will* become ill-conditioned as the number of degrees of freedom increase.

If the degrees of freedom of the element involve the derivatives of the function, we may adjust the above procedure. Given the value of the function $f_i$ and the fist derivative $f'_i$, for example, we may write:

$$f'(x) = \lfloor H'(x) \rfloor \{a\} = \lfloor 0 \quad 1 \quad 2x \quad \ldots \rfloor \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \end{Bmatrix}$$

Then the [T] matrix may be constructed such that:

$$[T] = \begin{bmatrix} \lfloor H(x_1) \rfloor \\ \lfloor H'(x_1) \rfloor \\ \lfloor H(x_2) \rfloor \\ \lfloor H'(x_2) \rfloor \\ \vdots \end{bmatrix}$$

In this case, the number of degrees of freedom per element will be equal to twice the number of nodes and the function $N_i(x)$ will appear in pairs. The first function of the pair will have a value of one at its corresponding node while the slope at that node is zero, and has a value and slope of zero at every other node. Meanwhile, the second function of the pair will have a slope of one and a value of zero at the corresponding node while its value and slope are equal to zero at every other node.

Note that, in both cases above, the order of the polynomial terms in the *H(x)* matrix does not affect the resulting shape functions because they are going to be rearranged using the *T* matrix.

### 2.2. Element Matrices

Each finite element problem will generate a set of matrices for each element. These matrices are derived from the physical model that is usually presented in the form of a differential equation. The matrices are constructed by integrating some derivative of the function that usually appears in the form:

$$k_e = \int_{x_1}^{x_2} Q(x) D^m(\lfloor N(x) \rfloor)^T D^m(\lfloor N(x) \rfloor) dx$$

Where *Q(x)* is some function that describes the physical properties of the problem, *D<sup>m</sup>(.)* is a differential operator that differentiates the function *f(x)* *m* times. For example, in mechanics of material, the finite element matrix for a bar in static problem may be obtained using:

$$k_e = \int_{x_1}^{x_2} E(x) A(x) \{N'(x)\} \lfloor N'(x) \rfloor dx$$

Where *E(x)* and *A(x)* are the modulus of elasticity and the cross-section area respectively. For simple problems, the above integral may be performed by hand or using symbolic manipulators. However, it is more convenient to use numerical integration in order to add flexibility to the programming. Nevertheless, some cases will not allow for hand or symbolic manipulators' integration, thus it is common to find numerical integration routines associated with any finite element program. One of the popular techniques used for numerical integration is the Gauss-Legendre quadrature which produces *almost* exact integration for polynomials that may reach up to 24th order. 12

### 2.3. 2-D Problems

In 2-D problems, the interpolation polynomials should include combinations of the *x* and *y* coordinates. To identify the terms, the common practice is to select them from what is known as the Pascal triangle (See Figure 1). The number of terms selected should be equal to the number of degrees of freedom and it is recommended that the terms should create a symmetric shape in that triangle.

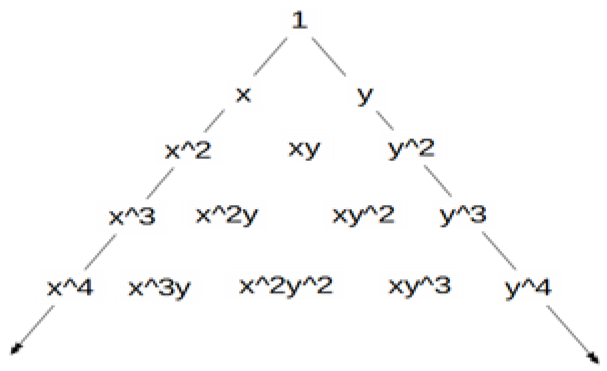

Figure 1. Pascal triangle

The Pascal triangle is quite handy in many problems, however, if you strict your work to using the full polynomials in both directions, then the generation of the terms becomes quite straight forward by multiplying both x and y-polynomials. For example, for a quadrilateral element with a single degree of freedom per node, we need a linear polynomial in each of the directions, hence, we may write:

$$H^*(x,y)=\begin{Bmatrix}1\\y\end{Bmatrix}\lfloor 1 \quad x\rfloor=\begin{bmatrix}1 & x\\ y & xy\end{bmatrix}$$

$$\rightarrow H(x,y)=\lfloor 1 \quad x \quad y \quad xy\rfloor$$

Then we proceed with the *H(x,y)* vector just as in the 1-D cases:

$$[T]=\begin{bmatrix}\lfloor H(x_1,y_1)\rfloor\\ \lfloor H(x_2,y_2)\rfloor\\ \vdots\end{bmatrix}$$

From which we obtain:

$$\lfloor N(x,y)\rfloor=\lfloor H(x,y)\rfloor[T^{-1}]$$

Note, again, that the order of the polynomial terms in the *H* vector will not affect the resulting shape functions similar to what we had in the 1-D problems. At this point, we still have both problems when generating the higher order elements, namely, the matrix inversion and the numerical integration. In the following section, we will attempt generating the matrices using exact integration to avoid the numerical integration problem.

## 3. EXACT INTEGRATION

To avoid the errors introduced by the numerical integration, we may resolve to exact integration using symbolic manipulators (wxMaxima® , Mathematica® , Mable® , etc …) but then we are back to the problem of rewriting the element matrices into our code, with all the problems that may include, or write the whole finite element model using the symbolic package which is normally extremely slow when it comes to numerical manipulations compare to numerical coding packages (Octave® , Matlab® , etc ...) or programming languages (Fortran, C++, etc …). Thus, we resolve to write an algorithm to perform the exact integration on the numerical package for specific elements with unknown number of degrees of freedom. This is where the *H(x)* row-matrix becomes very handy. Since the row-matrix is composed of simple polynomial terms, their differentiation, and later integration, is readily evaluated.

Let's examine the case of a bar element with *n* nodes (and *n* degrees of freedom). For the bar element, the element matrix, stiffness matrix, is evaluated using the integration:

$$k_e=\int_{x_1}^{x_2} EA\{N'(x)\}\lfloor N'(x)\rfloor dx$$

For the sake of the illustration, we will assume that the modulus of elasticity and the cross-section area are constants, thus, we may write:

$$k_e=EA\int_{x_1}^{x_2}[T^{-1}]^T\{H'(x)\}\lfloor H'(x)\rfloor[T^{-1}]dx$$

But, the transformation matrix is also constant, thus, it may be dragged out of the integration leaving us with the derivative of the *H(x)* vector.

$$k_e=EA[T^{-1}]^T\int_{x_1}^{x_2}\{H'(x)\}\lfloor H'(x)\rfloor dx[T^{-1}]$$

In this case, the *T* matrix may be given as:

$$[T]=\begin{bmatrix}\lfloor H(x_1)\rfloor\\ \lfloor H(x_2)\rfloor\\ \vdots\\ \lfloor H(x_n)\rfloor\end{bmatrix}$$

If we examine the terms of the *H(x)* vector, we may write:

$$\lfloor H(x)\rfloor=\lfloor 1 \quad x \quad x^2 \quad \ldots\rfloor \quad \text{or} \quad h_i=x^{i-1} \quad i=1,2,3,\ldots,n$$

Thus the first derivative may be written as:

$$\lfloor H'(x)\rfloor=\lfloor 0 \quad 1 \quad 2x \quad \ldots\rfloor$$

or

$$h'_i=\begin{matrix}0 & i=1\\ (i-1)x^{i-2} & i=2,3,\ldots,n\end{matrix}$$

Hence, we may write the matrix:

$$\{H'(x)\}\lfloor H'(x)\rfloor = [G(x)]$$

where

$$g_{ij} = h'_i h'_j = \begin{cases} 0 & \text{when } i \vee j = 1 \\ (i-1)(j-1)\,x^{(i-2)+(j-2)} & i,j = 2,3,\ldots,n \end{cases}$$

From that, we may get the integration:

$$k_e = EA[T^{-1}]^T \int_{x_1}^{x_2} [G(x)]dx[T^{-1}] = EA[T^{-1}]^T[G^*][T^{-1}]$$

where

$$g^*_{ij} = \begin{cases} 0 & \text{when } i \vee j = 1 \\ \dfrac{(i-1)(j-1)}{(i-2)+(j-2)+1}\left(x_2^{(i-2)+(j-2)+1} - x_1^{(i-2)+(j-2)+1}\right) & i,j = 2,3,\ldots,n \end{cases}$$

Similarly, for any 1-D problem involving any derivative, we may readily obtain the general term for the G matrix and perform the integration to obtain the G* matrix.

Let's examine the case of a beam element with *n* nodes (and *2n* degrees of freedom). For the beam element, the element matrix, stiffness matrix, is evaluated using the integration:

$$k_e = \int_{x_1}^{x_2} EI\{N''(x)\}\lfloor N''(x)\rfloor dx$$

For the sake of the illustration, we will assume that the modulus of elasticity and the cross-section second moments of area are constants, thus, we may write:

$$k_e = EI\int_{x_1}^{x_2} [T^{-1}]^T\{H''(x)\}\lfloor H''(x)\rfloor[T^{-1}]dx$$

But, the transformation matrix is also constant, thus, it may be dragged out of the integration leaving us with the derivative of the *H(x)* vector.

$$k_e = EI[T^{-1}]^T\int_{x_1}^{x_2} \{H''(x)\}\lfloor H''(x)\rfloor dx[T^{-1}]$$

In this case, the *T* matrix may be given as:

$$[T] = \begin{bmatrix} \lfloor H(x_1)\rfloor \\ \lfloor H'(x_1)\rfloor \\ \lfloor H(x_2)\rfloor \\ \lfloor H'(x_2)\rfloor \\ \vdots \\ \lfloor H(x_n)\rfloor \\ \lfloor H'(x_n)\rfloor \end{bmatrix}$$

If we examine the terms of the *H(x)* vector, we may write:

$$\lfloor H(x)\rfloor = \lfloor 1 \quad x \quad x^2 \quad \ldots\rfloor \quad \text{or} \quad h_i = x^{i-1} \quad i = 1,2,3,\ldots,n$$

Thus the second derivative may be written as:

$$\lfloor H''(x)\rfloor = \lfloor 0 \quad 0 \quad 2 \quad 6x \quad \ldots\rfloor$$

or

$$h''_i = \begin{cases} 0 & i = 1,2 \\ (i-1)(i-2)\,x^{i-3} & i = 3,4,\ldots,n \end{cases}$$

Thus, we may write the matrix:

$$\{H''(x)\}\lfloor H''(x)\rfloor = [G(x)]$$

where

$$g_{ij} = h''_i h''_j = \begin{matrix} 0 \quad \text{when } i \vee j = 1,2 \\ (i-1)(i-2)(j-1)(j-2)\,x^{(i-3)+(j-3)} \quad i,j = 3,4,\ldots,n \end{matrix}$$

From that, we may get the integration:

$$k_e = EI\,[T^{-1}]^T \int_{x_1}^{x_2} [G(x)]\,dx\,[T^{-1}] = EA\,[T^{-1}]^T [G^*][T^{-1}]$$

where

$$g^*_{ij} = \begin{matrix} 0 \text{ when } i \vee j = 1,2 \\ \dfrac{(i-1)(i-2)(j-1)(j-2)}{(i-3)+(j-3)+1}\left(x_2^{(i-3)+(j-3)+1} - x_1^{(i-3)+(j-3)+1}\right) \quad i,j = 3,4,\ldots,n \end{matrix}$$

For 2-D problems, the problem becomes a little more complex. The general term of the *H(x,y)* vector will depend on the order at which we select the terms from the Pascal's triangle. For illustration, let's assume that we are going to generate a rectangular element for plate bending that ensures continuity of slope ($C^1$ element). In this case, the number of nodes in the element may be selected to be $n^2$ where $n$ is the number of nodes per side of the element (n=2,3,4, …) in this case the number of degrees of freedom per element will be $4n^2$, and the x and y polynomials, each, will be of *2n-1* order. We may write:

$$[H^*(x,y)] = \begin{bmatrix} 1 & x & x^2 & \ldots & x^{2n-1} \\ y & xy & x^2 y & \ldots & x^{2n-1} y \\ \vdots & . & . & \ldots & \vdots \\ y^{2n-1} & x\,y^{2n-1} & x^2\,y^{2n-1} & \ldots & x^{2n-1}\,y^{2n-1} \end{bmatrix}$$

Thus, we may write the *H(x,y)* vector as:

$$\lfloor H(x,y) \rfloor = \lfloor 1 \quad x \quad \ldots \quad x^{2n-1} \quad y \quad xy \quad \ldots \quad x^{2n-1} y^{2n-1} \rfloor$$

Whose general term may be given by:

$$h_k = h^*_{ij} = x^{j-1} y^{i-1} \text{ where } k = j + 2n(i-1), \quad i,j = 1,2,\ldots,2n$$

And the derivatives of the *H* vector, may be written as:

$$h_{k,xx} = (j-1)(j-2)\,x^{j-3} y^{i-1}$$
$$h_{k,yy} = (i-1)(i-2)\,x^{j-1} y^{i-3}$$
$$h_{k,xy} = (j-1)(i-1)\,x^{j-2} y^{i-2}$$
$$k = j + 2n(i-1)$$

The element stiffness matrix for the static loading problem of thin plates may be written as:

$$k_e = \int_{x_1}^{x_2}\int_{y_1}^{y_2} [T^{-1}]^T [C]^T [Q][C][T^{-1}]\,dy\,dx$$

Where:

$$[T]_{4n^2\times 4n^2}=\begin{bmatrix}\lfloor H(x_1,y_1)\rfloor\\ \lfloor H_x(x_1,y_1)\rfloor\\ \lfloor H_y(x_1,y_1)\rfloor\\ \lfloor H_{xy}(x_1,y_1)\rfloor\\ \lfloor H(x_2,y_2)\rfloor\\ \lfloor H_x(x_2,y_2)\rfloor\\ \lfloor H_y(x_2,y_2)\rfloor\\ \lfloor H_{xy}(x_2,y_2)\rfloor\\ \vdots\\ \lfloor H(x_{n^2},y_{n^2})\rfloor\\ \lfloor H_x(x_{n^2},y_{n^2})\rfloor\\ \lfloor H_y(x_{n^2},y_{n^2})\rfloor\\ \lfloor H_{xy}(x_{n^2},y_{n^2})\rfloor\end{bmatrix}$$

And:

$$[C]_{3\times 4n^2}=\begin{bmatrix}\lfloor H_{xx}(x,y)\rfloor\\ \lfloor H_{yy}(x,y)\rfloor\\ 2\lfloor H_{xy}(x,y)\rfloor\end{bmatrix}$$

Whose general term may be written as:

$$c_{1k}=h_{k,xx}$$
$$c_{2k}=h_{k,yy}$$
$$c_{3k}=h_{k,xy}$$
$$k=1,2,\ldots,4n^2$$

*[Q]* is the plate stiffness which may be written, for the sake of compactness, in the form:

$$[Q]=\begin{bmatrix}a & b & 0\\ b & a & 0\\ 0 & 0 & c\end{bmatrix}$$

The matrix *[G]* may be written as:

$$[G]=[C]^T[Q][C]$$

where the general term may evaluated by:

$$g_{kl}=a(h_{k,xx}h_{l,xx}+h_{k,yy}h_{l,yy})+b(h_{k,yy}h_{l,xx}+h_{k,xx}h_{l,yy})+4ch_{k,xy}h_{l,xy}$$

Let's now examine each term.

$$h_{k,xx}h_{l,xx}=(j_k-1)(j_k-2)(j_l-1)(j_l-2)x^{j_k+j_l-6}y^{i_k+i_l-2}$$
$$h_{k,yy}h_{l,xx}=(i_k-1)(i_k-2)(j_l-1)(j_l-2)x^{j_k+j_l-4}y^{i_k+i_l-4}$$
$$h_{k,xx}h_{l,yy}=(j_k-1)(j_k-2)(i_l-1)(i_l-2)x^{j_k+j_l-4}y^{i_k+i_l-4}$$
$$h_{k,yy}h_{l,yy}=(i_k-1)(i_k-2)(i_l-1)(i_l-2)x^{j_k+j_l-2}y^{i_k+i_l-6}$$
$$h_{k,xy}h_{l,xy}=(j_k-1)(i_k-1)(j_l-1)(i_l-1)x^{j_k+j_l-4}y^{i_k+i_l-4}$$

Where

$$j_k = mod\left(\frac{k-1}{2n+1}\right)+1$$

$$i_k = \frac{k-j_k}{2n+1}+1$$

$$j_l = mod\left(\frac{l-1}{2n+1}\right)+1$$

$$i_l = \frac{l-j_l}{2n+1}+1$$

Performing the integration, we get:

$$\alpha_1=\int_{x_1}^{x_2}\int_{y_1}^{y_2} h_{k,xx}h_{l,xx}\,dy\,dx=\frac{(j_k-1)(j_k-2)(j_l-1)(j_l-2)}{(j_k+j_l-5)(i_k+i_l-1)}(x_2^{j_k+j_l-5}-x_1^{j_k+j_l-5})(y_2^{i_k+i_l-1}-y_1^{i_k+i_l-1})$$

Similarly:

$$\alpha_2=\int_{x_1}^{x_2}\int_{y_1}^{y_2} h_{k,yy}h_{l,xx}\,dy\,dx=\frac{(i_k-1)(i_k-2)(j_l-1)(j_l-2)}{(j_k+j_l-3)(i_k+i_l-3)}(x_2^{j_k+j_l-3}-x_1^{j_k+j_l-3})(y_2^{i_k+i_l-3}-y_1^{i_k+i_l-3})$$

$$\alpha_2=\int_{x_1}^{x_2}\int_{y_1}^{y_2} h_{k,yy}h_{l,xx}\,dy\,dx=\frac{(i_k-1)(i_k-2)(j_l-1)(j_l-2)}{(j_k+j_l-3)(i_k+i_l-3)}(x_2^{j_k+j_l-3}-x_1^{j_k+j_l-3})(y_2^{i_k+i_l-3}-y_1^{i_k+i_l-3})$$

$$\alpha_4=\int_{x_1}^{x_2}\int_{y_1}^{y_2} h_{k,yy}h_{l,yy}\,dy\,dx=\frac{(i_k-1)(i_k-2)(i_l-1)(i_l-2)}{(j_k+j_l-1)(i_k+i_l-5)}(x_2^{j_k+j_l-1}-x_1^{j_k+j_l-1})(y_2^{i_k+i_l-5}-y_1^{i_k+i_l-5})$$

$$\alpha_5=\int_{x_1}^{x_2}\int_{y_1}^{y_2} h_{k,xy}h_{l,xy}\,dy\,dx=\frac{(j_k-1)(i_k-1)(j_l-1)(i_l-1)}{(j_k+j_l-3)(i_k+i_l-3)}(x_2^{j_k+j_l-3}-x_1^{j_k+j_l-3})(y_2^{i_k+i_l-3}-y_1^{i_k+i_l-3})$$

From which, we may get the integral of the general term of the *[G]* matrix in the form:

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2} g_{kl}\,dy\,dx=g_{kl}^{*}=a(\alpha_1+\alpha_4)+b(\alpha_2+\alpha_3)+4C\alpha_5$$

In both the 1-D and 2-D problems illustrated above, we were able to create the element matrices exactly using the trick of integrating the *H* vector and its derivatives. However, the limiting condition on generating high order elements will always be the inversion of the *T* matrix which will become ill-conditioned with higher order elements. For example, the plate bending element described above worked only until the number of nodes became 16 (64 DOF) (7th order polynomial in each of the x and y directions) using double precision numbers on the Octave numerical manipulator. Hence, a search for another technique was required.

## 4. LAGRANGE POLYNOMIALS

### 4.1. 1-D problems

The Lagrange interpolation polynomial for an *(n-1)*th order polynomial, which may be used when *n* degrees of freedom are give, may be written in the form:

$$f(x)=\sum_{i=1}^{n} f_i\prod_{\substack{j=1\\ j\neq i}}^{n-1}\frac{x-x_j}{x_i-x_j}=\lfloor N_1(x)\quad N_2(x)\quad \dots\quad N_n(x)\rfloor\begin{Bmatrix}f_1\\ f_2\\ \vdots\\ f_n\end{Bmatrix}$$

Where:

$$N_i(x)=\prod_{\substack{j=1\\ j\neq i}}^{n-1}\frac{x-x_j}{x_i-x_j}$$

Thus, we may obtain the first derivative in the form:

$$N'_i(x)=\sum_{\substack{k=1\\ k\neq i}}^{n-1}\frac{1}{x_i-x_k}\prod_{\substack{j=1\\ j\neq k\\ j\neq i}}^{n-1}\frac{x-x_j}{x_i-x_j}$$

We may also obtain further derivatives for use in the element equation, however, let's remember that the function interpolation is based on the values of the function, thus, if we need to have any of the derivatives of the function as a degree of freedom, we *cannot* use this interpolation method. Nevertheless, we have obtained the shape functions without having to go through the matrix inversion problem described in the previous section. On the other hand, performing the exact integration will not be readily available for the general term since it now involves multiplication of several linear terms. Luckily, as mentioned earlier, the Gauss quadrature integration provides high degree of accuracy for a relatively high order polynomial integration.

Note that he order of the Lagrange polynomials used above has to coincide with the order of the node numbering unlike the cases we described in sections 2 and 3 above.

### 4.2. 2-D Problems

With the same advantages and disadvantages of the 1-D problem, the derivation of the shape functions for a 2-D problem comes straight forward by multiplying the polynomials in the x and y directions.

$$f(x,y)=\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} f_{ij}\prod_{\substack{k=1\\ k\neq i}}^{n_x-1}\frac{x-x_k}{x_i-x_k}\prod_{\substack{l=1\\ l\neq j}}^{n_y-1}\frac{y-y_l}{y_j-y_l}=\lfloor N_{11}(x,y)\quad N_{12}(x,y)\quad\dots\quad N_{n_xn_y}(x,y)\rfloor\begin{Bmatrix}f_{11}\\ f_{12}\\ \vdots\\ f_{n_xn_y}\end{Bmatrix}$$

In a vector, we usually use a single index to point to the different element, thus, we need to create a relation between the vector index and the i and j counters of the nodes counters in a 2-D element. A simple relation for counting the nodes in the x-direction first would be in the form:

$$m=i+(j-1)n_x$$
$$i=1,2,\dots,n_x$$
$$j=1,2,\dots,n_y$$
$$m=1,2,\dots,n_x\times n_y$$

Thus we may write the general shape function in the form:

$$N_m(x,y)=N_{ij}(x,y)=\prod_{\substack{k=1\\ k\neq i}}^{n_x-1}\frac{x-x_k}{x_i-x_k}\prod_{\substack{l=1\\ l\neq j}}^{n_y-1}\frac{y-y_l}{y_j-y_l}$$

For which we may evaluate the first derivatives to be:

$$N_{m,x}(x,y)=N_{ij,x}(x,y)=\sum_{\substack{p=1\\ p\neq i}}^{n_x-1}\frac{1}{x_i-x_p}\prod_{\substack{k=1\\ k\neq p\\ k\neq i}}^{n_x-1}\frac{x-x_k}{x_i-x_k}\prod_{\substack{l=1\\ l\neq j}}^{n_y-1}\frac{y-y_l}{y_j-y_l}$$

$$N_{m,y}(x,y)=N_{ij,y}(x,y)=\prod_{\substack{k=1\\ k\neq i}}^{n_x-1}\frac{x-x_k}{x_i-x_k}\sum_{\substack{q=1\\ q\neq j}}^{n_y-1}\frac{1}{y_j-y_q}\prod_{\substack{l=1\\ l\neq q\\ l\neq j}}^{n_y-1}\frac{y-y_l}{y_j-y_l}$$

$$N_{m,xy}(x,y)=N_{ij,xy}(x,y)=\sum_{\substack{p=1\\ p\neq i}}^{n_x-1}\frac{1}{x_i-x_p}\prod_{\substack{k=1\\ k\neq p\\ k\neq i}}^{n_x-1}\frac{x-x_k}{x_i-x_k}\sum_{\substack{q=1\\ q\neq j}}^{n_y-1}\frac{1}{y_j-y_q}\prod_{\substack{l=1\\ l\neq q\\ l\neq j}}^{n_y-1}\frac{y-y_l}{y_j-y_l}$$

## 5. MODIFYING THE INTERPOLATION FUNCTION

Borrowing from the *Spline* interpolation techniques, we may work around with the Lagrange polynomials to obtain polynomials that may include derivatives of the function. In the following, we will work with functions that are required to satisfy the value of the function and its first derivative at each node of the element. We will start be deriving the cubic polynomial that will fit 2 values and two slopes, then, we will describe the process for quintic polynomial, followed by the general approach for obtaining the *(2n-1)*th polynomial that may be used for n-node elements.

### 5.1. The Polynomials

Using the similarities we obtained in the 3rd and 5th order polynomial derivation, we will present a generalization in the following section. We may write the general relation for a function of the *(2n-1)*th order that uses n pairs of polynomials as:

$$f(x)=\sum_{i=1}^{n} S_i(x)$$

Where:

$$S_i(x)=(C_1+C_2(x-x_i))\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

This function needs to satisfy the conditions:

$$S_i(x_i)=f_i$$
$$S'_i(x_i)=f'_i$$

For the first condition, we get:

$$C_1=f_i$$

To proceed with the second condition, we get the slopes as:

$$S'_i(x)=C_2\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+2(C_1+C_2(x-x_i))\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{x-x_k}{(x_i-x_k)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq k}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

Substituting:

$$S'_i(x_i)=f'_i=C_2+2C_1\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}$$

Which gives:

$$C_2=f'_i-2f_i\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}$$

Thus we may write the pair of polynomials as:

$$S_i(x)=\left(f_i+\left(f'_i-2f_i\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}\right)(x-x_i)\right)\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

Rearranging, we obtain:

$$S_i(x)=f_i\left(1-2(x-x_i)\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+f'_i(x-x_i)\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

From which we may write the shape functions as:

$$N_{i,1}(x)=\left(1-2(x-x_i)\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$N_{i,2}(x)=(x-x_i)\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$i=1,2,\ldots,n$$

In 2-D problems, the interpolation functions will be a straight forward generalization of the above procedure giving the function in the form:

$$f(x,y)=\sum_{m=1}^{n} S_m(x,y)$$
$$n=n_x\times n_y$$

Where:

$$S_m(x)=(C_1+C_2(x-x_i)+C_3(y-y_j)+C_4(x-x_i)(y-y_j))\prod_{\substack{k=1\\k\neq i}}^{n_x}\left(\frac{x-x_k}{x_i-x_k}\right)^2\prod_{\substack{l=1\\l\neq j}}^{n_y}\left(\frac{y-y_l}{y_j-y_l}\right)^2$$
$$m=i+(j-1)*n_x$$
$$i=1,2,\ldots,n_x$$
$$j=1,2,\ldots,n_y$$
$$m=1,2,\ldots,n$$

This function needs to satisfy the conditions:

$$S_m(x_i,y_j)=f_{ij}$$
$$S_{m,x}(x_i,y_j)=f_{ij,x}$$
$$S_{m,y}(x_i,y_j)=f_{ij,y}$$
$$S_{m,xy}(x_i,y_j)=f_{ij,xy}$$

Which will result in a quartet of shape functions in the form:

$$N_{m,1}(x,y)=\left(1-2(x-x_i)\sum_{\substack{k=1\\k\neq i}}^{n_x}\frac{1}{x_i-x_k}\right)\left(1-2(y-y_j)\sum_{\substack{p=1\\p\neq j}}^{n_y}\frac{1}{y_j-y_p}\right)\prod_{\substack{l=1\\l\neq i}}^{n_x}\left(\frac{x-x_l}{x_i-x_l}\right)^2\prod_{\substack{q=1\\q\neq j}}^{n_y}\left(\frac{y-y_q}{y_j-y_q}\right)^2$$

$$N_{m,2}(x,y)=(x-x_i)\left(1-2(y-y_j)\sum_{\substack{p=1\\p\neq j}}^{n_y}\frac{1}{y_j-y_p}\right)\prod_{\substack{l=1\\l\neq i}}^{n_x}\left(\frac{x-x_l}{x_i-x_l}\right)^2\prod_{\substack{q=1\\q\neq j}}^{n_y}\left(\frac{y-y_q}{y_j-y_q}\right)^2$$

$$N_{m,3}(x,y)=\left(1-2(x-x_i)\sum_{\substack{k=1\\k\neq i}}^{n_x}\frac{1}{x_i-x_k}\right)(y-y_j)\prod_{\substack{l=1\\l\neq i}}^{n_x}\left(\frac{x-x_l}{x_i-x_l}\right)^2\prod_{\substack{q=1\\q\neq j}}^{n_y}\left(\frac{y-y_q}{y_j-y_q}\right)^2$$

$$N_{m,4}(x)=(x-x_i)(y-y_j)\prod_{\substack{l=1\\l\neq i}}^{n_x}\left(\frac{x-x_l}{x_i-x_l}\right)^2\prod_{\substack{q=1\\q\neq j}}^{n_y}\left(\frac{y-y_q}{y_j-y_q}\right)^2$$
$$i=1,2,\ldots,n_x$$
$$j=1,2,\ldots,n_y$$
$$m=1,2,\ldots,n_x\times n_y$$

## 5.2. The Derivatives

In the problems that make use of the shape functions described in the previous section, the finite element model usually requires the first and second derivatives of the shape function to perform the required calculations. In this section, we will present those derivatives for the 1-D problem without elaborations.

$$N_{i,1}(x)=\left(1-2(x-x_i)\sum_{\substack{k=1\\k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\prod_{\substack{j=1\\j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$N_{i,2}(x)=(x-x_i)\prod_{\substack{j=1\\j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$N'_{i,1}(x)=\left(-2\sum_{\substack{k=1\\k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\prod_{\substack{j=1\\j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+\left(1-2(x-x_i)\sum_{\substack{k=1\\k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\sum_{\substack{l=1\\l\neq i}}^{n}2\frac{x-x_l}{(x_i-x_l)^2}\prod_{\substack{j=1\\j\neq i\\j\neq l}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$N'_{i,2}(x)=\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+(x-x_i)\sum_{\substack{k=1\\ k\neq i}}^{n}2\frac{x-x_k}{(x_i-x_k)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq k}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$N''_{i,1}(x)=2\left(-2\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\sum_{\substack{l=1\\ l\neq i}}^{n}2\frac{x-x_l}{(x_i-x_l)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq l}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$+\left(1-2(x-x_i)\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{1}{x_i-x_k}\right)\left(\sum_{\substack{l=1\\ l\neq i}}^{n}\frac{2}{(x_i-x_l)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq l}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+\sum_{\substack{l=1\\ l\neq i}}^{n}2\frac{x-x_l}{(x_i-x_l)^2}\sum_{\substack{m=1\\ m\neq i\\ m\neq l}}^{n}2\frac{x-x_m}{(x_i-x_m)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq l\\ j\neq m}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2\right)$$

$$N''_{i,2}(x)=2\sum_{\substack{k=1\\ k\neq i}}^{n}2\frac{x-x_k}{(x_i-x_k)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq k}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2$$

$$+(x-x_i)\left(\sum_{\substack{k=1\\ k\neq i}}^{n}\frac{2}{(x_i-x_k)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq k}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2+\sum_{\substack{k=1\\ k\neq i}}^{n}2\frac{x-x_k}{(x_i-x_k)^2}\sum_{\substack{l=1\\ l\neq i\\ l\neq k}}^{n}2\frac{x-x_l}{(x_i-x_l)^2}\prod_{\substack{j=1\\ j\neq i\\ j\neq k\\ j\neq l}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2\right)$$

## 6. NUMERICAL RESULTS

All the test cases used were confined to generating the element stiffness matrix for different problems for isoparametric $C^0$ and $C^1$ continuous elements. The number of nodes were increased and the Eigenvalues of the matrix were evaluated until the lowest (negative) Eigenvalue reached an absolute values larger than $10^{-4}$ at which the element was considered a failure since the matrices investigated were supposed to be positive semi-definite. The results for the different problems are presented below.

Table 1 presents the change of the values of the minimum Eigenvalue of the stiffness matrix generated for a bar element. If we ignore the numbers that have an absolute value less than $10^{-10}$ considering them as a numerical zero, we may still find that increasing the number of nodes per element start to introduce negative Eigenvalues until we reach the failure criterion set in this work as $10^{-4}$. Note that, unpredictably, the exact integration method failed before the numerical integration at 15 nodes per element. On the other hand, the Lagrange polynomial method, which does not use the transformation matrix, was able to produce satisfactory results up to 23 nodes per element which approaches the limit of the 12 point numerical integration technique used.

Table 2 presents the results obtained for a beam element. Again we find that the exact integration and numerical integration elements, both, failed around the 18 and 16 DOF respectively which coincides with the number of DOF at which failure occurred in the bar element. Meanwhile, the Modified Lagrange method was able to continue all the way up to 24 DOF which reaches the limit imposed by the numerical integrator.

Table 1. Change of the lowest Eigenvalue for the Bar problem using different methods

| Number of Nodes | Classical Method | Exact Integration | Lagrange Polynomials |
|---|---|---|---|
| 2 | 0.0000 | 0.0000 | 0.0000 |
| 5 | 0.0000 | $1.3550 * 10^{-15}$ | $-3.2092 * 10^{-14}$ |
| 13 | $-4.3771 * 10^{-11}$ | $1.8269 * 10^{-8}$ | $-4.7237 * 10^{-9}$ |
| 14 | $-3.5311 * 10^{-10}$ | $1.2446 * 10^{-6}$ | $-8.3456 * 10^{-9}$ |
| 15 | $-6.7339 * 10^{-9}$ | $-7.2761 * 10^{3}$ - Failure | $-1.7247 * 10^{-8}$ |
| 16 | $-4.0141 * 10^{-5}$ | | $-6.8237 * 10^{-8}$ |
| 17 | -0.0063877 - Failure | | $-2.7224 * 10^{-7}$ |
| 24 | | | -0.000284 – Almost! |

Table 2. Change of the lowest Eigenvalue for the Beam problem using different methods

| Number of Nodes (DOF) | Classical Method | Exact Integration | Modified Lagrange Polynomials |
|---|---|---|---|
| 2 (4) | $-1.8447 * 10^{-15}$ | $-1.6821 * 10^{-15}$ | $-4.0007 * 10^{-16}$ |
| 5 (10) | $-2.7423 * 10^{-10}$ | $-2.3674 * 10^{-10}$ | $-7.2653 * 10^{-13}$ |
| 6 (12) | $-2.7096 * 10^{-8}$ | $-4.4350 * 10^{-10}$ | $-3.3600 * 10^{-10}$ |
| 7 (14) | $9.5001 * 10^{-8}$ | $4.6870 * 10^{-8}$ | $-1.4082 * 10^{-9}$ |
| 8 (16) | -2,4339 - Failed | $-2.7862 * 10^{-4}$ | $-1,0945 * 10^{-9}$ |
| 9 (18) | | -0.045658 - Failed | $-3.2284 * 10^{-8}$ |
| 11 (22) | | | $-7.7276 * 10^{-7}$ |
| 12 (24) | | | $-2.9915 * 10^{-5}$ |

Table 3. Change of the lowest Eigenvalue for the Plate problem using different methods

| Number of Nodes (DOF) | Classical Method | Exact Integration | Modified Lagrange Polynomials |
|---|---|---|---|
| 4 (16) | $-2.4664 * 10^{-16}$ | $-9.9632 * 10^{-16}$ | $-4.2013 * 10^{-15}$ |
| 9 (36) | $-4.0203 * 10^{-13}$ | $-4.3150 * 10^{-13}$ | $-1.1133 * 10^{-14}$ |
| 16 (64) | $-2.1111 * 10^{-9}$ | $-3.6665 * 10^{-10}$ | $9.6377 * 10^{-16}$ |
| 25 (100) | $-2.6320 * 10^{+5}$ - Failed | $-1.9562 * 10^{+7}$ - Failed | $-2.7317 * 10^{-14}$ |
| 100 (400) | | | $-4.5663 * 10^{-6}$ |
| 121 (484) | | | $-2.1076 * 10^{-4}$ – Almost! |
| 144 (576) | | | -0.023264 - Failed |

Table 3 presents the results for a plate bending problem. The same patterns observed in the bar and beam elements can be observed in the plate problem for the modified Lagrange case.

## 7. CONCLUSIONS

Several problems were examined in this work to demonstrate the limitations imposed on the creation of *super elements* with high number of nodes and degrees of freedom. The element matrices were generated using simple, Lagrange, and modified Lagrange polynomials and the integration was performed numerically using Gauss-Lagrange quadrature or exact when possible. Problems involving 1-D elements with 1 or 2 DOF per node and 2-D problems with 4 DOF per node were examined by increasing the number of nodes per element until the lowest (negative) Eigen value of the stiffness matrix had an absolute value greater than $10^{-4}$. From the results obtained above we may conclude the following:

- Automated generation of higher order isoperimetric $C^0$ and $C^1$ continuous element model was enabled using an open source numerical manipulator (Octave®) using the different methods presented in this work.
- Generating higher order elements is computationally expensive compared to lower ones. Thus, the h-version is more efficient for the same accuracy than the p-version.
- Using numerical integration to evaluate the element matrices is efficient and accurate enough for most practical purposes
- Using Lagrange and modified Lagrange polynomial increases the stability of the results for higher order element problems when they become a necessity to use. In such cases, the limitation on the results' accuracy will be imposed by the accuracy of the numerical integration scheme.

Further work with the modified Lagrange polynomials, proposed in this work, is needed to enable the inclusion of higher order derivatives in the interpolation polynomials. Such research will help in the generation of elements with the needed derivatives without falling into the trap of the ill-conditioned transformation matrix.

## REFERENCES AND BIBLIOGRAPHY

1. J. N. Reddy, *An Introduction to the Finite Element Method*, 3rd ed., McGraw Hill, 2006

2. M. Tawfik, *Finite Element Analysis*, book draft, ResearchGate.net, 2017, DOI: 10.13140/RG.2.2.32391.70560

3. M. Tawfik, *A Spectral Finite Element Model for Thin Plate Vibration*, International Congress on Sound and Vibration (ICSV14), Cairns, Australia, 9-12 July 2007. DOI: 10.13140/2.1.1123.6167

4. M. Tawfik, *Higher Order and Spectral Finite Element Model for Thin Plate Vibration – Much Difference?*, 5th International Conference on Mathematics and Information Sciences (ICMIS2016), Zewail City of Science and Technology 11-13- Feb. 2016

5. I. Babuska, B. Szabo, and I. Katz, *The p-Version of the Finite Element Method*, SIAM Journal of Numerical Analysis, Vol. 18, No. 33, pp. 515-545, June 1981

6. L. Demkpvicz, J. Kurtz, D. Pardo, M. Paszynski, W. Rachowicz, and A. Zdunek, *Computing with hp-Adaptive Finite Elements*, Chapman and Hall, 2008

7. J. Korelc and P. Wriggers, *Automation of Finite Element Methods*, Springer Verlag, 2016, DOI 10.1007/978-3-319-39005-5

8. A. Bespalov and N. Heuer, *A New H(div) Conforming p-Interpolation Operator in Two Dimensions*, ESAIM: Mathematical Modeling and Numerical Analysis, Vol. 45, pp. 255-275, 2011, DOI:10.1051/m2an/2010039

9. M. Tawfik, *Fundamentals of Numerical Analysis*, book draft, ResearchGate.net, 2017, DOI: 10.13140/RG.2.2.25680.81925

10. S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 5th ed., McGraw Hill, 2006

11. L. Demkowicz, J. Gopalakrishnan, and J. Schöberl, *Polynomial Extension Operators. P art I*, Mathematics and Statistics Faculty Publications and Presentations. 61, 2008 http://pdxscholar .library .pdx.edu/mth_fac/61

12. M. Costable, M. Dauge, and L. Demkovicz, *Polynomial Extension Operators for H1, H(curl), H(div) – Spaces on a Cube*, Mathematics of Computation, Vol. 77, No. 264, pp. 1967-1999, 2008

13. D. Zwillinger (editor), *Standard Mathematical Tables and Formulae*, 30th ed., CRC Press, 2000