Computation of electrostatic forces by the virtual work method

Simone Riva

September 13, 2007

Abstract

The most popular methods for computing the force acting on a body in a finite elements context are the Maxwell stress tensor, the virtual work and the eggshell methods. In this project we study and compare these three methods analytically and numerically. We derive the algorithm of the methods and we compare the procedures to proof the equivalence between them. In the second part of the project we develop some procedure for building a more efficient *shell* for improving the efficiency of the original procedures.

Contents

List of Figures

List of Tables

1

Introduction

1.1. Electric potential

In a capacitor the electrostatic potential u is a solution of the Laplace equation with the Dirichlet boundary conditions. In this project we normally use a configuration of the domain Ω with an inner body $\partial\mathbf{\Omega_1}$ with a Dirichlet potential $f_1(\mathbf{x})$ and an external boundary $\partial\mathbf{\Omega_2}$ set to zero.

$$\nabla^2 u = 0 \tag{1.1}$$

$$u\,|_{\partial\mathbf{\Omega_1}} = f_1(\mathbf{x}) \tag{1.2}$$

$$u\,|_{\partial\mathbf{\Omega_2}} = 0 \tag{1.3}$$

The two boundary $\partial\mathbf{\Omega_1}$ and $\partial\mathbf{\Omega_2}$ can have any arbitrary shape. In some case we add a second inner body.

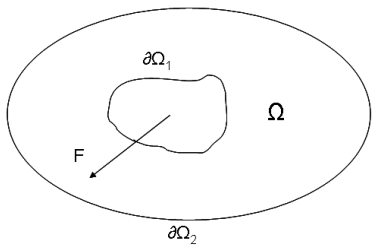


Figure 1.1.: The generic problem configuration

1.1.1. Computing the force

The computation of the electrostatic force in a finite elements context has been discussed in several papers. Normally they describe the different methods separately. In this work we have done a direct comparison between the three and most popular methods.

We have worked with some two dimensional problems. With the linear finite elements and triangular meshes.

Given the FEM solution computing the force acting on a movable rigid object $\partial\Omega_1$ can be performed in several ways. The most classical method is the *Maxwell stress tensor*. This method consists in integrating the Maxwell stress tensor over a surface S around the body. The Maxwell stress tensor procedure it is easy to implements but the reliability of the result is low and also the convergence rate is not very good. Thus some people have developed news and more powerful procedures. The second procedure is the *virtual work method*, described by J.L. Coulomb in his paper [Cou83]. The method is based on application of virtual work principle. The virtual work method is equivalent to the Maxwell stress tensor method in his basic implementation. But can be easily extended with a dramatic improvement of the results. The third method is the *eggshell* procedure described for the fist time by F. Hernotte [Hen04]. The main idea, also in this case, is based on the virtual shifting of the object in a vector field. From the work expression we straightforwardly derive the expression for the force. Also this method offers a very good flexibility.

2

Procedures for computig the force

2.1. Maxwell Stress Tensor

The total force acting on a rigid body may be obtained by integrating the Maxwell Stress Tensor over an arbitrary surface S in the empty space that include only the rigid object $\partial\mathbf{\Omega_1}$.

$$\mathbf{F} = \epsilon \int_S \mathbf{M}\mathbf{n} d\Omega \tag{2.1}$$

where $\mathbf{n}$ is the outer *unit normal* vector of the surface S and $\mathbf{M}$ is the Maxwell Stress Tensor on the electrostatic potential that is defined as follow:

$$\mathbf{M} := \frac{-1}{2} (\mathbf{E} \cdot \mathbf{E}) \mathbf{I} + \mathbf{E}\mathbf{E}^T \tag{2.2}$$

thus, if we expand the preview formula, we have:

$$\mathbf{F} = \epsilon \int_S (\mathbf{E}\mathbf{n})\mathbf{E} - \frac{1}{2}\mathbf{E}^2\mathbf{n} dS \tag{2.3}$$

Witch is the Maxwell Stress Tensor formula for computing the force.

In an electrostatic potential u the electric field $\mathbf{E}$ can be directly computed with the gradient of u:

$$\mathbf{E} = -\nabla u \tag{2.4}$$

The Maxwell stress tensor procedure has the following advantages:

1. Only one computation of the potential u is needed.
2. It is easy to compute the surface integral given a linear FEM solution.
3. The surface S can be arbitrary chosen in the empty space.
4. It is a very fast procedure.

But has the following disadvantages:

1. The convergence rate of the solution is slow: for obtaining a good result we must perform a lot on refinement.
2. The algorithm for building the surrounding surface S in some cases it is not easy (concave body or very irregulars meshes).
3. In a finite element context optimum choice of S depends on the type of elements. Thus the integration is not surface independent.
4. In a FEM context the shape of the optimal surface S depend on the mesh, so the quality of the solution is strongly influenced from the mesh structure.

In the next sections we describe the *virtual work* and the *eggshell* methods, that own all the advantage of the *Maxwell stress tensor* and solve the disadvantages.

2.2. Virtual Work Method

The *virtual work procedure* for computing the force $\mathbf{F}$ is based on the concept, from the basic physics, that the force is the gradient of the work. So if we perform a virtual translation of the rigid object $\partial\mathbf{\Omega_1}$ in the electric potential it is possible to compute the force $\mathbf{F}$.

The electrostatic coenergy is given by

$$W = \frac{\epsilon}{2}\int_{\Omega_V} |\mathbf{E}|^2 d\Omega_V \tag{2.5}$$

where $\mathbf{E}$ is the electric field (2.4) and $\Omega_V \subseteq \Omega$ is the virtually distorted empty area (or shell) around $\partial\mathbf{\Omega_1}$. We use the shell as measure of the virtual displacement.

The global force acting on the movable rigid object $\partial\mathbf{\Omega_1}$ in the $\mathbf{p}$ direction can be calculated by performing the derivative of the coenergy along the virtual direction $\mathbf{p}$

$$\mathbf{F_p} = \frac{\partial}{\partial\mathbf{p}} W \tag{2.6}$$

The resulting force is the projection of the global force $\mathbf{F}$ on the vector $\mathbf{p}$. For computing the global force we must compute the force $\mathbf{F_p}$ in two orthogonal directions (normally x and y).

If we combine the formulae 2.6 with 2.5, one can write:

$$\mathbf{F_p} = \frac{\epsilon}{2}\frac{\partial}{\partial \mathbf{p}}\int_{\Omega_V} |\mathbf{E}|^2 d\Omega_V \tag{2.7}$$

Finally by using the chain rule we can write the preview expression like this

$$\mathbf{F_p} = \epsilon \int_{\Omega_V} \mathbf{E}\frac{\partial}{\partial \mathbf{p}}\mathbf{E} d\Omega + \frac{\epsilon}{2}\int_{\Omega_V} |\mathbf{E}|^2 \frac{\partial}{\partial \mathbf{p}}\left(d\Omega_V\right) \tag{2.8}$$

which is the *virtual work expression* for the force. This integral will be solved in the FEM space in the section 2.6.

2.3. EggShell Method

The *eggshell* method has been described for the first time by F. Hernotte in the paper [Hen04]. In this we section we give an intuitive idea of the *egg shell* approach, you can find the full algebraic derivation in the paper of F. Hernotte [Hen04]. The main idea of the *egg shell* method is very similar to the one of *virtual work*. It is based on the idea to computing the change of the energy (the work) in an electrostatic field by moving the body $\partial\mathbf{\Omega_1}$.

The work $\partial\mathbf{\Omega_1}$ received by the body in an electric field is the change of the energy $\dot{W}$ if we move the rigid body $\partial\mathbf{\Omega_1}$ with a given velocity.

As firs we define the vector field $\mathbf{v}$ that describe the virtual shifting of $\partial\mathbf{\Omega_1}$.

The time derivative of W (see also [Hen04]) can be written as follow:

$$\frac{\partial W}{\partial t} = \dot{W} = \int_{\Omega_S} \mathbf{M} : \nabla\mathbf{v} d\Omega_S \tag{2.9}$$

where : is the tensor product $A : B = \sum_{ij} A_{ij}B_{ij}$. If we perform the integration by part (by using the rules described in [Hen04]) we can write 2.9 as follow

$$\dot{W} = -\int_{\Omega_S} (div\mathbf{M}) \cdot \mathbf{v} d\Omega_S + \int_{\partial\mathbf{\Omega_1}} \mathbf{n}\mathbf{M}\mathbf{v}\ d\left(\partial\mathbf{\Omega_1}\right) \tag{2.10}$$

where $\mathbf{n}$ denote the outward normal of $\partial\mathbf{\Omega_1}$. Now from this formula it is possible to derive an expression for computing the force if we shift the body of an infinitesimal amount $\delta\mathbf{u}$. For computing the time derivative of the energy W we must define the vector field $\mathbf{v}$ as a function of $\delta\mathbf{u}$. We describe the deformation $\mathbf{v}$ only in a surrounding empty area $\Omega_S \subseteq \Omega$ (the shell) around $\partial\mathbf{\Omega_1}$ with a smooth function γ whose values is 1 on the boundary of $\partial\mathbf{\Omega_1}$ and 0 on the outer surface of the shell. So we can write:

$$\mathbf{v} = \gamma\delta\dot{\mathbf{u}} \qquad \nabla\mathbf{v} = \nabla\gamma\delta\dot{\mathbf{u}} \tag{2.11}$$

Now, from the elementary physics, we can write the work received from $\partial\mathbf{\Omega_1}$ as the multiplication between the force on $\partial\mathbf{\Omega_1}$ and the shifting $\mathbf{v}$. So the expression 2.10 become:

$$\dot{W} = -\mathbf{F}\delta\dot{\mathbf{u}} = \int_{\Omega_S} \mathbf{M} : \nabla\mathbf{v} d\Omega_S \tag{2.12}$$

Finally if we simplify the term $\delta\dot{\mathbf{u}}$ (for more details see [Hen04]) we have the expression for computing the force acting on $\mathbf{F}$:

$$\mathbf{F} = -\int_{\Omega_S} \mathbf{M}\nabla\gamma d\Omega_S \tag{2.13}$$

Which is the *egg shell* formula for computing the force on $\partial\mathbf{\Omega_1}$. Only the Maxwell stress tensor of empty space is required here.

2.4. Finite Elements settings

In this project we have used the triangular linear finite elements in a 2D domain. But the work described in this report can be easily implemented with the quadrilateral finite elements or in 3D with tetraedical elements.
In a finite elements context the approximate solution u_N of the laplace equation is a weighted sum of the local shapes functions λ_i:

$$u_N = \sum_i \mu_i \lambda_i \tag{2.14}$$

Where the shapes functions λ_i are the *barycentric coordinates functions* on each triangular element $K = (\mathbf{a}^1; \mathbf{a}^2; \mathbf{a}^3)$, they are defined as follow:

$$\lambda_1(\mathbf{x}) = \frac{1}{2A_K}\left(\mathbf{x} - \begin{pmatrix} a_1^2 \\ a_2^2 \end{pmatrix}\right) \cdot \begin{pmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{pmatrix} \tag{2.15}$$

$$\lambda_2(\mathbf{x}) = \frac{1}{2A_K}\left(\mathbf{x} - \begin{pmatrix} a_1^3 \\ a_2^3 \end{pmatrix}\right) \cdot \begin{pmatrix} a_2^3 - a_2^1 \\ a_1^2 - a_1^3 \end{pmatrix} \tag{2.16}$$

$$\lambda_3(\mathbf{x}) = \frac{1}{2A_K}\left(\mathbf{x} - \begin{pmatrix} a_1^1 \\ a_2^1 \end{pmatrix}\right) \cdot \begin{pmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{pmatrix} \tag{2.17}$$

where A_K is the area of the element K and a_j^i are the vertices of the triangular element (see also figure 2.6).

The gradient of the FEM solution is directly derived from the gradients of the barycentric coordinates functions. Then the gradient of u_N is:

$$\nabla u_N = \sum_i \mu_i \nabla \lambda_i \tag{2.18}$$

therefore the electric field $\mathbf{E}$ in the FEM space is $\mathbf{E} = -\nabla u_N$. Where the gradients of the barycentric coordinate functions are:

$$\nabla \lambda_1 = \frac{1}{2A_K} \begin{pmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{pmatrix} \tag{2.19}$$

$$\nabla \lambda_2 = \frac{1}{2A_K} \begin{pmatrix} a_2^3 - a_2^1 \\ a_1^2 - a_1^3 \end{pmatrix} \tag{2.20}$$

$$\nabla \lambda_3 = \frac{1}{2A_K} \begin{pmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{pmatrix} \tag{2.21}$$

In the virtual work implementation, we need a mapping function $\Phi(\tilde{\mathbf{x}}) : \mathbb{R}^2 \to \mathbb{R}^2$ between the reference element $\tilde{K}$ and the global element K.

The mapping can be directly defined with the barycentric coordinates function:

$$\Phi(\tilde{\mathbf{x}}) = \sum_{i=1}^{3} \tilde{\lambda}_i(\tilde{\mathbf{x}}) \mathbf{a}^{\mathbf{i}} \tag{2.22}$$

Where $\mathbf{a}^{\mathbf{i}}$ are the vertices coordinates of the global element, $\tilde{\lambda}_i$ are the barycentric coordinates function on the reference element and $\tilde{\mathbf{x}}$ is a point in the reference mesh.

2.5. Maxwell Stress Tensor implementation

The Maxwell stress tensor can be easily implemented in a FEM context by performing a piecewise integration on the closed surface S around $\partial \mathbf{\Omega_1}$.

With the triangular mesh the optimum result is obtained when S cross the triangle on the middle of the edges.

The generic procedure is the following:

1. Generate the surface S.

2. On each segment compute the local contribution to the force.

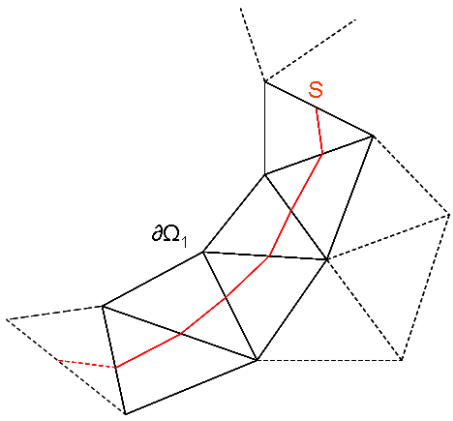


Figure 2.1.: A portion of a surrounding surface S in a FEM context

2.5.1. Generating the surface S

The optimal surface cross the edges in the middle point of the segments. So we need one segment on each element.

The simplest way for building the surface is computing the middle points on the segments and sorting these points in counter clockwise order respect to the center of the object.

This algorithm is very simple but unstable if the mesh is irregular or the object $\partial\mathbf{\Omega_1}$ is concave.

2.5.2. Itegration

Given the segment on an element the Maxwell Stress Tensor can be straightforwardly integrated. With the linear FEM the gradient of u_N is constant on the element. The normal of the surface can be computed with the rotation of the segment. On each element we have:

$$\mathbf{E}_e = -\nabla u_e = -\sum_{i=1}^{3} \mu_i \nabla \lambda_i \tag{2.23}$$

which is clearly constant on the element.

If we have the two midpoints P_1 and P_2 in counter clockwise order the normal $\mathbf{n}_e$ on the surface can be computed by rotating of $-\pi/2$ the segment:

$$\mathbf{n}_e = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \frac{(P_2 - P_1)}{|P_2 - P_1|} \tag{2.24}$$

Thus on each element the integral can be easily computed by multiplying the integrand by the distance between P_1 and P_2. So the local contribution to the force is:

$$\mathbf{F}_e = \epsilon |P_2 - P_1| \cdot \left((\mathbf{E}_e \mathbf{n}_e)\, \mathbf{n}_e - \frac{1}{2} \mathbf{E}_e^2 \mathbf{n}_e \right) \tag{2.25}$$

The global force is the sum over the local forces: $\mathbf{F}_N = \sum_e \mathbf{F}_e$

2.6. Virtual Work Implementation on a triangular mesh

In this section we derive the analytical integration of 2.8 on a triangular element. This method can be easily derived also for quadrilateral meshes or in 3D cases.

The generic algorithm for computing the force with the virtual work principle in a finite element context is the following:

1. Compute the solution of the problem with the finite element procedure.
2. Compute the derivative of the coordinates of the virtual displaced nodes.
3. Compute the volume integral of the virtual distorted area (2.27).

Given the finite elements solution, the first problem is computing the derivative of the displaced nodes. This derivative can be expressed on each node $[x_i, y_i]$ in the finite elements domain. For example if we are interested, to the derivative of $[x_i, y_i]$ versus $\mathbf{x}$ and $\mathbf{y}$ directions we have:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{p}_x} x_i = p_i \quad \frac{\partial}{\partial \mathbf{p}_y} x_i = 0 \\ \frac{\partial}{\partial \mathbf{p}_x} y_i = 0 \quad \frac{\partial}{\partial \mathbf{p}_y} y_i = p_i \end{aligned} \tag{2.26}$$

where $p_i = 1$ if the node lie on the boudary $\partial\mathbf{\Omega_1}$ of the virtual moved rigid object. p_i can be 0 or rescaled ($0 \leq p < 1$) with some interpolation procedures on the virtual distorted empty space of the domain Ω_M (see also the chapter 4) and must be 0 on the others boundary ($\partial\mathbf{\Omega_2}$...). In a practical implementation of the virtual work procedure we assign to each node the value p_i. So for the triangle $K = (a^1; a^2; a^3)$ we have the values $(p_1; p_2; p_3)$.
The procedure can be also generalized to all directions $\mathbf{p}$ but we must known the mapping function between the original domain and the virtual distorted domain.

Given the virtual distorted area we compute the volume integral (2.8) by summing up the local contribution to the force of each elements that lie in the distorted volume. So we have:

$$\mathbf{F_N} = \sum_{K_e \in \mathcal{M}_D} \left[\epsilon \int_{K_e} \mathbf{E}_e \frac{\partial}{\partial \mathbf{p}} \mathbf{E}_e dK_e + \frac{\epsilon}{2} \int_{K_e} |\mathbf{E}_e|^2 \frac{\partial}{\partial \mathbf{p}} (dK_e) \right] \tag{2.27}$$

where $\mathcal{M}_D$ is the set of the virtually distorded elements and $\mathbf{E}_e$ is the local electic field on a single element:

$$\mathbf{E}_e = \begin{pmatrix} E_x \\ E_y \end{pmatrix} \tag{2.28}$$

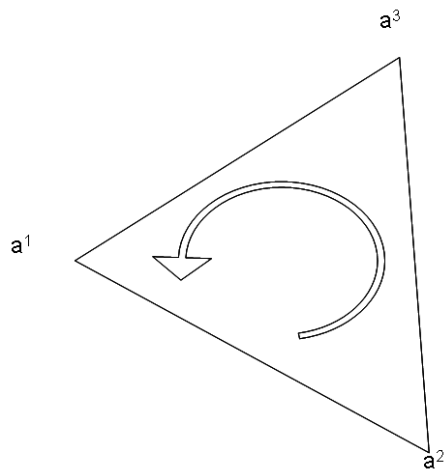


Figure 2.2.: An element where: $a^1 = (a_1^1\ a_2^1)^\top\ a^2 = (a_1^2\ a_2^2)^\top\ a^3 = (a_1^3\ a_2^3)^\top$

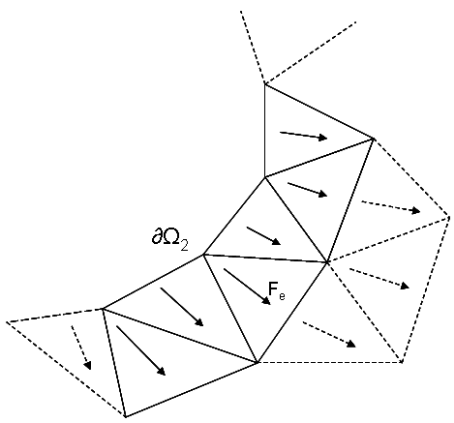


Figure 2.3.: A portion of the virtual distorted area. We compute the local cotribution to the force on each element

Now for implement the *virtual work* principle in a finite element context we work with a reference element on the coordinates system $[v, w]$. The transformation between the reference coordinates and the global coordinates is calculated with a *Mapping* function $\Phi(\tilde{\mathbf{x}})$. Where $\Phi(\tilde{\mathbf{x}})$ is:

$$\Phi(\tilde{\mathbf{x}}) := \sum_{i=1}^{3} \lambda_i\,(v, w)\,\mathbf{a}^i \tag{2.29}$$

Where $\lambda_i\,(v, w)$ are the local shape functions on the reference element and $\mathbf{a^i}$ are the vertices coordinates of the meshes expressed in the global coordinates. In this case, we use the barycentric coordinates functions.

Now we have the means to implement the Virtual Work method in a FEM context.
The integral over an element is expressed in term of reference coordinates:

$$\int_K f(\mathbf{x})d\mathbf{x} = \int_{\tilde{K}} f(\Phi(\tilde{\mathbf{x}}))|G|d\tilde{\mathbf{x}} \tag{2.30}$$

where $\tilde{K}$ is the reference element and $|G|$ is the determinant of the Jacobian matrix of the mapping function $\Phi(\tilde{\mathbf{x}})$:

$$|G| = \det\left((D\Phi(\tilde{\mathbf{x}}))\right) \tag{2.31}$$

From the gradient transformation formula, the electric field $\mathbf{E}_e$ on the element K_e can be also written according to the mapping function:

$$\mathbf{E}_e = -\nabla \sum_{i=1}^{3} \mu_i \lambda_i = -G^{-\top} \sum_{i=1}^{3} \mu_i \nabla_{\tilde{\mathbf{x}}} \tilde{\lambda}_i(\tilde{\mathbf{x}}) \tag{2.32}$$

Now the derivative of $\mathbf{E}$ versus $\mathbf{p}$ become (by using the derivative of the identity $G^\top \cdot G^{-\top} = I$):

$$\frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} = -G^{-\top} \frac{\partial G^\top}{\partial \mathbf{p}} G^{-\top} \sum_{i=1}^{3} \mu_i \nabla_{\tilde{\mathbf{x}}} \tilde{\lambda}_i(\tilde{\mathbf{x}}) \tag{2.33}$$

From the equations 2.32 and 2.33 we can derive the derivative of $\mathbf{E}_e$:

$$\frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} = G^{-\top} \frac{\partial G^\top}{\partial \mathbf{p}} \nabla u_e \tag{2.34}$$

where $\nabla u_e = (\partial_x u_e \;\; \partial_y u_e)^\top$ is the gradinet on the element.

Now we define

$$\frac{\partial}{\partial \mathbf{p}}(dK_e) = \frac{\partial |G|}{\partial \mathbf{p}} |G|^{-1} dK_e \tag{2.35}$$

Finally the integration for the local force expression 2.27 (on a single element) is:

$$\mathbf{F}_e = \epsilon A_K \left(\mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} + \frac{1}{2} \mathbf{E}_e^2 \frac{\partial |G|}{\partial \mathbf{p}} |G|^{-1} \right) \tag{2.36}$$

where A_K is the area of the element K_e.

For completing the integration of the virtual work, we must derive the Jacobian matrix G of the mapping function $\Phi(\tilde{\mathbf{x}})$, that can be easily derived using the the gradient of the shapes functions λ_i (see also 2.19 2.20 2.21).

$$G^\top = \begin{pmatrix} a_1^2 - a_1^1 & a_2^2 - a_2^1 \\ a_1^3 - a_1^1 & a_2^3 - a_2^1 \end{pmatrix} \equiv \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{2.37}$$

For computing the derivative of $G^\top$ we use the values p_i assigned to each vertex a^i_j. As described in the formula 2.26.

The derivative versus the x direction of $G^\top$ is

$$\frac{\partial G^\top}{\mathbf{p}_x} = \begin{pmatrix} p_2 - p_1 & 0 \\ p_3 - p_1 & 0 \end{pmatrix} \tag{2.38}$$

and likewise the derivative of $G^\top$ versus the y direction is

$$\frac{\partial G^\top}{\mathbf{p}_y} = \begin{pmatrix} 0 & p_2 - p_1 \\ 0 & p_3 - p_1 \end{pmatrix} \tag{2.39}$$

finally the derivative of the determinat of the Jacobian $|G| = ad - cb$ can be easily computed with the common derivations rules. The derivative of $|G|$ versus an arbitrary direction $\mathbf{p}$ is:

$$\frac{\partial |G|}{\partial \mathbf{p}} = \frac{\partial a}{\partial \mathbf{p}} d + a \frac{\partial d}{\partial \mathbf{p}} - \frac{\partial c}{\partial \mathbf{p}} b - c \frac{\partial b}{\partial \mathbf{p}} \tag{2.40}$$

if we are interested to the derivatives of $|G|$ versus x we have:

$$\frac{\partial |G|}{\partial \mathbf{p}_x} = (p_2 - p_1)d - (p_3 - p_1)b \tag{2.41}$$

and the derivative of $|G|$ versus y is

$$\frac{\partial |G|}{\partial \mathbf{p}_y} = a(p_3 - p_1) - c(p_2 - p_1) \tag{2.42}$$

Now we can compute the local contribution to the force of each element with 2.27. The total force, acting on $\partial \mathbf{\Omega_1}$, is calculated by adding all local forces contribution.

2.7. Eggshell implementation

The implementation of the *eggshell* method it is very easy. We must simply define the gradient of γ, the Maxwell stress tensor and integrating over the element.

At first we define a smooth function γ on the element. The easiest way is using the barycentric coordinates function. Thus γ on each triangular element can be written:

$$\gamma = \sum_{i=1}^{3} r_i \lambda_i(\mathbf{x}) \tag{2.43}$$

where λ_i are the baricentric coordinates funtions (see also equations 2.15, 2.16 and 2.17) and $0 \leq r_i \leq 1$ is an arbitrary value that describe the shell Ω_S (see also the chapter 2.3).

The gradient of γ can be found by using the gradient of λ_i (see the formulae 2.19, 2.20 and 2.21):

$$\nabla\gamma = \sum_{i=1}^{3} r_i \nabla\lambda_i(\mathbf{x}) \tag{2.44}$$

The Maxwell stress tensor on a 2D FEM context $\mathbf{M}$ can be found with the definition of $\mathbf{E}$ (2.23).

$$\mathbf{M}_e = \begin{pmatrix} E_x^2 - \frac{1}{2}\mathbf{E}^2 & E_x E_y \\ E_x E_y & E_y^2 - \frac{1}{2}\mathbf{E}^2 \end{pmatrix} \tag{2.45}$$

finally the local contribution to the force is:

$$\mathbf{F}_e = -A_K \mathbf{M} \nabla\gamma(\mathbf{x}) \tag{2.46}$$

which Is the eggshell formula 2.13 integrated over an element.

2. Procedures for computig the force

3

Equivalence between methods

After the first tests we have observed that the *virtual work* method, with the *one on the boundary* interpolation and the Maxwell stress tensor returns always the same results. In a second time we have also observed that the *eggshell* and the *virtual work* methods returns the same results if the *shells* are equivalents. Thus we have decided to proof analytically the equivalence between the three methods in a FEM context.

In the followings proofs we start from the main formula and we develop and simpilify the expressions up to the equivalence between the methods is manifest.

3.1. Equivalence between Maxwell Stress Tensor and Virtual Work

3.1.1. Virtual Work

Now, we consider the case of a triangular element $K_e = (\mathbf{a}^1; \mathbf{a}^2; \mathbf{a}^3)$ with the verticies $(\mathbf{a}^1; \mathbf{a}^2$ that lie on the boudary $\partial\mathbf{\Omega_1}$. In this case we consider the *one on the boudary* interpolation, thus the *virtual displacements derivatives* p on the verticies are:
1 on the verticies $\mathbf{a}^1$ and $\mathbf{a}^2$.
0 on the vetex $\mathbf{a}^3$.

The translated jacobian $G^\top$ of the mapping function $\Phi(\tilde{\mathbf{x}})$ is:

$$G^\top = \begin{pmatrix} a_1^2 - a_1^1 & a_2^2 - a_2^1 \\ a_1^3 - a_1^1 & a_2^3 - a_2^1 \end{pmatrix} \equiv \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3.1}$$

and the inverse of the jacobian $G^{-\top}$

$$G^{-\top} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{3.2}$$

The derivatives of $G^\top$ versus the x and y directions can be calculated by using the analytical derivatives 2.38, 2.39 and the values of p_i on the element (defined in this section).

Then the derivative of $G^\top$ versus x is

$$\frac{\partial G^\top}{\partial \mathbf{p}_x} = \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \tag{3.3}$$

and versus y is

$$\frac{\partial G^\top}{\partial \mathbf{p}_y} = \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.4}$$

The area A_K of the element can be calculated using the definition of cross product and the coordinates of the vertices of the element (see also 2.6):

$$A_K = \frac{(a_1^2 - a_1^1)(a_2^3 - a_2^1) - (a_2^2 - a_2^1)(a_1^3 - a_1^1)}{2} \equiv \frac{ad - bc}{2} \tag{3.5}$$

If we assume that $\epsilon = 1$ the local contribution to the force is

$$\mathbf{F}_e = A_K \left(\mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} + \frac{1}{2} \mathbf{E}_e^2 \frac{\partial |G|}{\partial \mathbf{p}} |G|^{-1} \right) \tag{3.6}$$

First part

Now we develop the first part of the formula 3.6: $A_K \mathbf{E}_e \dfrac{\partial \mathbf{E}_e}{\partial \mathbf{p}}$ as a function of the x and y directions.

By using the formula 2.34 we can write

$$A_K \mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} = A_K \mathbf{E}_e G^{-\top} \frac{\partial G^\top}{\partial \mathbf{p}} \nabla u_e \tag{3.7}$$

where $\nabla u_e = (\partial_x u_e \;\; \partial_y u_e)^\top$ is the gradient of u_N in an element.

If we replace the definition of 3.2 in the preview formula we have

$$A_K \mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} = \frac{A_K}{ad - bc} \mathbf{E}_e \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \frac{\partial G^\top}{\partial \mathbf{p}} \nabla u_e \tag{3.8}$$

By using the the definition of A_K (3.7) it can be simplified to

$$A_K \mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}} = \frac{1}{2} \mathbf{E}_e \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \frac{\partial G^\top}{\partial \mathbf{p}} \nabla u_e \tag{3.9}$$

To obtain the expressions as a function of the x and y directions we use the for:derGynum and for:derGynum.

Thus with a pair of simple steps we derive the two expression for the x direction:

$$A_K \mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}_x} = \frac{1}{2} \left[(a_2^1 - a_2^2)(\partial_x u_e)^2 + (a_1^2 - a_1^1)(\partial_x u_e)(\partial_y u_e) \right] \tag{3.10}$$

and also for y

$$A_K \mathbf{E}_e \frac{\partial \mathbf{E}_e}{\partial \mathbf{p}_y} = \frac{1}{2} \left[(a_2^1 - a_2^2)(\partial_x u_e)(\partial_y u_e) + (a_1^2 - a_1^1)(\partial_y u_e)^2 \right] \tag{3.11}$$

Second part

The second part of the expression 3.6: $\frac{1}{2} \mathbf{E}_e^2 \frac{\partial |G|}{\partial \mathbf{p}} |G|^{-1}$ can be easly derived as follow:

At first, we must derive the derivatives of the determinant of the Jacobian G versus x and y.

$$|G| = ad - cb \tag{3.12}$$

the derivatives of $|G|$ versus the x and y directions can be found by using the formula 2.42 and the values of p_i on the element:

$$\frac{\partial |G|}{\partial \mathbf{p}_x} = b \tag{3.13}$$

$$\frac{\partial |G|}{\partial \mathbf{p}_y} = -a \tag{3.14}$$

so the expression in x and y diretions becomes:

$$\frac{A_K}{2}\mathbf{E}_e^2|G|^{-1}\frac{\partial|G|}{\partial\mathbf{p}_x} = \frac{\mathbf{E}_e^2}{4}(a_2^2 - a_2^1) \tag{3.15}$$

$$\frac{A_K}{2}\mathbf{E}_e^2|G|^{-1}\frac{\partial|G|}{\partial\mathbf{p}_y} = \frac{\mathbf{E}_e^2}{4}(a_1^1 - a_1^2) \tag{3.16}$$

Result

if we collect the preview results of the first and second part in a single formula we obtain the full expression for the local force:

$$F_x = \frac{1}{2}\left[(a_2^1 - a_2^2)(\partial_x u_e)^2 + (a_1^2 - a_1^1)(\partial_x u_e)(\partial_y u_e)\right] + \frac{\mathbf{E}_e^2}{4}(a_2^2 - a_2^1) \tag{3.17}$$

$$F_y = \frac{1}{2}\left[(a_2^1 - a_2^2)(\partial_x u_e)(\partial_y u_e) + (a_1^2 - a_1^1)(\partial_y u_e)^2\right] + \frac{\mathbf{E}_e^2}{4}(a_1^1 - a_1^2) \tag{3.18}$$

Witch is the simplified expression for computing the local contribution to the force with the virtual work method and the one on the boudary interpolation. Note that this expression is a particular case of the full expression derived in section 3.2.1.

3.1.2. Maxwell Stress Tensor

In this section we analize the integrtion of the maxwell stress tensor on an element $K = (\mathbf{a}^1; \mathbf{a}^2; \mathbf{a}^2)$ where $\mathbf{a}^1$ and $\mathbf{a}^2$ lie on the boudary and $\mathbf{a}^3$ is free. The segment $\overline{\mathbf{P}_1\mathbf{P}_2}$ is a portion of the surface S.

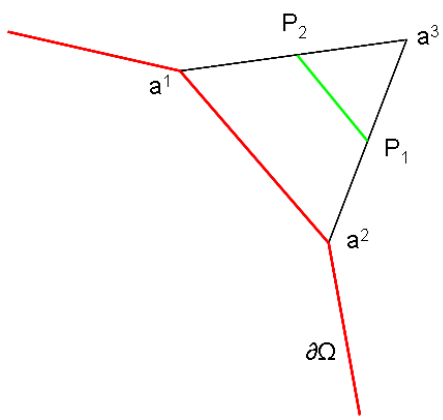


Figure 3.1.: The element configuration used in this chapter

Given the middles points $\mathbf{P}_1$ and $\mathbf{P}_2$ of the edges $(\mathbf{a}^1; \mathbf{a}^3)$ and $(\mathbf{a}^2; \mathbf{a}^3)$. The distance between $\mathbf{P}_1$ and $\mathbf{P}_2$ is $d = |\mathbf{P}_2 - \mathbf{P}_1|$.

The integration of the maxwell stress tensor along the segment is siply:

$$\mathbf{F}_e = d \cdot \left((\mathbf{E}_e \mathbf{n}_e) \, \mathbf{E}_e - \frac{1}{2} \mathbf{E}_e^2 \mathbf{n}_e \right) \tag{3.19}$$

The unit normal $\mathbf{n}$ on the surface segment is:

$$\mathbf{n}_e = \frac{1}{2d} \begin{pmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{pmatrix} \tag{3.20}$$

If we expand and observe the first part of the equation 3.19:

$$(\mathbf{E}_e \cdot \mathbf{n}_e) \, \mathbf{E}_e = \frac{1}{2} \left(\partial_x u_e (a_2^1 - a_2^2) + \partial_y u_e (a_1^2 - a_1^1) \right) \begin{pmatrix} \partial_x u_e \\ \partial_y u_e \end{pmatrix} \tag{3.21}$$

we immediately conclude that is equivalent to the equation 3.10 and 3.11.

By expandinng the send part of the maxwell stress tensor equation we have:

$$\frac{1}{2} \mathbf{E}_e^2 \mathbf{n}_e = \frac{1}{4} \mathbf{E}_e^2 \begin{pmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{pmatrix} = -\frac{1}{4} \mathbf{E}_e^2 \begin{pmatrix} a_2^2 - a_2^1 \\ a_1^1 - a_1^2 \end{pmatrix} \tag{3.22}$$

thus if we collect the partial results, the resulting locals forces in x and y direction are:

$$F_x = \frac{1}{2} \left[(a_2^1 - a_2^2)(\partial_x u_e)^2 + (a_1^2 - a_1^1)(\partial_x u_e)(\partial_y u_e) \right] + \frac{\mathbf{E}_e^2}{4} (a_2^2 - a_2^1) \tag{3.23}$$

$$F_y = \frac{1}{2} \left[(a_2^1 - a_2^2)(\partial_x u_e)(\partial_y u_e) + (a_1^2 - a_1^1)(\partial_y u_e)^2 \right] + \frac{\mathbf{E}_e^2}{4} (a_1^1 - a_1^2) \tag{3.24}$$

that they are clearly equvalent to the *virtial work* expressions (3.17 3.18).

We can conclude that the *Virtual work procedure* and the *Maxewell Stress Tensor procedure* are equvalents.

3.1.3. Proof the equvalence between the different mesh configurations

In a real world application we must also consider all the possible configurations of the element $K = (\mathbf{a}^1; \mathbf{a}^2; \mathbf{a}^2)$:

The first case is where 2 nodes lie on the boundary but the enumeration of the vertices is different. For example $\mathbf{a}^3$ and $\mathbf{a}^1$ lie on the boundary and $\mathbf{a}^2$ is free. In this case we simply switch

the names of the vertices and the proof is the same. Note that these different enumerations produces different derivatives of the Jacobians matrix G if we use the formalism proposed in the preview section.

The second is the case where only one vertex lie on the boudary. For example $\mathbf{a}^1$ lie on the boudary and $\mathbf{a}^2$, $\mathbf{a}^3$ are free. In the maxwell stress tenson procedure the unit normal $\mathbf{n}$ to the segment become:

$$\mathbf{n}_e = \frac{1}{2d}\begin{pmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{pmatrix} \tag{3.25}$$

In the *virtual work* case the derivatives of the Jacobian matrix are:

$$\frac{\partial G^\top}{\partial \mathbf{p}_x} = \begin{pmatrix} -1 & 0 \\ -1 & 0 \end{pmatrix} \tag{3.26}$$

$$\frac{\partial G^\top}{\partial \mathbf{p}_y} = \begin{pmatrix} 0 & -1 \\ 0 & -1 \end{pmatrix} \tag{3.27}$$

For completing the proof, we need for the matrix product $G^{-\top}\dfrac{\partial G^\top}{\partial \mathbf{p}}$ in x and y directions and the derivatives of the $|G|$. Then we immediately observe the equivalence of the new formulae.

3.2. Equivalence between EggShell and Virtual Work

In this section, we proof that the virtual work procedure and the eggshell procedure are equivalent if the shell (or the virtually distorted area) is the same.

we must consider that the shell around $\partial\mathbf{\Omega_1}$ can have an arbitrary shape involving a lot of layers around the body. Thus, the development of the equations must be generic.

For obtaining this result, we have used a generic element K: Given a triangular mesh $K = (\mathbf{a}^1; \mathbf{a}^2; \mathbf{a}^3)$ we assign an arbitrary value $0 \leq r_i \leq 1$ to each vertex of K then for each element we have the triplet $(r_1; r_2; r_3)$.

3.2.1. Virtual Work

In this case, we use the values of r_i as a *virtual displacement derivative* ($r_i \equiv p_i$).

We define the Jacobian matrix as:

$$G^\top = \begin{pmatrix} a_1^2 - a_1^1 & a_2^2 - a_2^1 \\ a_1^3 - a_1^1 & a_2^3 - a_2^1 \end{pmatrix} \equiv \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3.28}$$

and his translated inverse:

$$G^{-\top} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{3.29}$$

The derivatives of $G^\top$ versus the x direction become

$$\frac{\partial G^\top}{\partial \mathbf{p}_x} = \begin{pmatrix} a_x & 0 \\ c_x & 0 \end{pmatrix} \equiv \begin{pmatrix} r_2 - r_1 & 0 \\ r_3 - r_1 & 0 \end{pmatrix} \tag{3.30}$$

and the derivative versus y

$$\frac{\partial G^\top}{\partial \mathbf{p}_y} = \begin{pmatrix} 0 & b_y \\ 0 & d_y \end{pmatrix} \equiv \begin{pmatrix} 0 & r_2 - r_1 \\ 0 & r_3 - r_1 \end{pmatrix} \tag{3.31}$$

First part

now, by using the definition of the element area A_K (3.5) and the equations 2.34, 3.30. The first part of the virtual work expression 3.6 derived versus x become:

$$\begin{aligned} A_K \mathbf{E}_e \frac{\partial \mathbf{E}}{\partial \mathbf{p}_x} &= A_K \mathbf{E}_e G^{-\top} \frac{\partial G^\top}{\partial \mathbf{p}_x} && (3.32) \\ &= \mathbf{E}_e \left(\frac{A_K}{ad - bc} \begin{pmatrix} a_x d - c_x b & 0 \\ -c a_x + a c_x & 0 \end{pmatrix} \nabla u_e \right) && (3.33) \\ &= \mathbf{E}_e \left(\frac{1}{2} \begin{pmatrix} a_x d - c_x b & 0 \\ -c a_x + a c_x & 0 \end{pmatrix} \nabla u_e \right) && (3.34) \end{aligned}$$

and the derivative versus y is:

$$
\begin{aligned}
A_K \mathbf{E}_e \frac{\partial \mathbf{E}}{\partial \mathbf{p}_y} &= A_K \mathbf{E}_e G^{-\top} \frac{\partial G^\top}{\partial \mathbf{p}_y} && (3.35)\\
&= \mathbf{E}_e \left(\frac{A_K}{ad - bc} \begin{pmatrix} 0 & b_y d - d_y b \\ 0 & -cb_y + ad_y \end{pmatrix} \nabla u_e \right) && (3.36)\\
&= \mathbf{E}_e \left(\frac{1}{2} \begin{pmatrix} 0 & b_y d - d_y b \\ 0 & -cb_y + ad_y \end{pmatrix} \nabla u_e \right) && (3.37)
\end{aligned}
$$

second part

For developing the second part of the virtual work expression 3.6: we must define the derivatives of $|G|$:

Given the derivatives 2.38 2.39, at first we define the derivative of $|G|$ versus x:

$$\frac{\partial |G|}{\partial \mathbf{p}_x} = a_x d - c_x b \tag{3.38}$$

and the derivative of $|G|$ versus y

$$\frac{\partial |G|}{\partial \mathbf{p}_y} = a d_y - c b_y \tag{3.39}$$

so the expression as a funtion of x become

$$\frac{1}{2} A_K \mathbf{E}_e^2 |G|^{-1} \frac{\partial |G|}{\partial \mathbf{p}_x} = \frac{1}{4} \mathbf{E}_e^2 (a_x d - c_x b) \tag{3.40}$$

and respect to y

$$\frac{1}{2} A_K \mathbf{E}_e^2 |G|^{-1} \frac{\partial |G|}{\partial \mathbf{p}_y} = \frac{1}{4} \mathbf{E}_e^2 (a d_y - c b_y) \tag{3.41}$$

resulting force

Finally we can collect the first and the second part of the expression.

If we consider that $\nabla u_e = -\mathbf{E}_e$ the resulting force computed with the wirtual work method is:

$$F_x = \frac{1}{2} \left[E_x^2 (-a_x d + c_x b) + E_x E_y (c a_x - a c_x) \right] + \frac{1}{4} \mathbf{E}_e^2 (a_x d - c_x b) \tag{3.42}$$

$$F_y = \frac{1}{2}\left[E_x E_y(-b_y d + d_y b) + E_y^2(cb_y - ad_y)\right] + \frac{1}{4}\mathbf{E}_e^2(ad_y - cb_y) \tag{3.43}$$

For simplifying the comparison with the eggsell formulae (3.52, 3.52) we reorganize the equation in this way:

$$F_x = (a_x d - c_x b)\left(\frac{1}{4}\mathbf{E}_e^2 - \frac{1}{2}E_x^2\right) + \frac{1}{2}E_x E_y(ca_x - ac_x) \tag{3.44}$$

$$F_y = (ad_y - cb_y)\left(\frac{1}{4}\mathbf{E}_e^2 - \frac{1}{2}E_y^2\right) + \frac{1}{2}E_x E_y(-b_y d + d_y b) \tag{3.45}$$

3.2.2. EggShell

The local contribution to the force in the eggshell method is:

$$\mathbf{F}_i = -\int_K M\nabla\gamma dx \tag{3.46}$$

where γ is a function over K defined as follow:

$$\gamma = \sum_{i=1}^{3} r_i \lambda_i(\mathbf{x}) \tag{3.47}$$

where λ_i are the baricentric coordinates functions (2.15 2.16 2.17)

the analitical integration over the mesh K become:

$$\mathbf{F}_e = -A_K M\nabla\gamma \tag{3.48}$$

and the gradinet of γ can be easly found with the gradients of the *baricentric coordinates functions* on K (see also 2.19 2.20 2.21). We define: $\nabla\gamma = (\partial_x\gamma \;\; \partial_y\gamma)^\top$.

The maxwell stress tensor M is

$$\mathbf{M} = \begin{pmatrix} E_x^2 - \frac{1}{2}\mathbf{E}^2 & E_x E_y \\ E_x E_y & E_y^2 - \frac{1}{2}\mathbf{E}_e^2 \end{pmatrix} \tag{3.49}$$

then the force computed with the eggshell method become:

The x component of the force is:

$$F_x = A_K\left\{-\left[\partial_x\gamma E_x^2 + \partial_y\gamma E_x E_y\right] + \frac{1}{2}\partial_x\gamma\mathbf{E}^2\right\} \tag{3.50}$$

and the y compunent:

$$F_y = A_K \left\{ - \left[\partial_x \gamma E_x E_y + \partial_y \gamma E_y^2 \right] + \frac{1}{2} \partial_y \gamma \mathbf{E}^2 \right\} \tag{3.51}$$

Now, as done in the previous section, we symplify the expression in the following way:

The x component:

$$F_x = A_K \left[\partial_x \gamma \left(\frac{1}{2} \mathbf{E}^2 - E_x^2 \right) - \partial_y \gamma E_x E_y \right] \tag{3.52}$$

and the y component

$$F_y = A_K \left[\partial_y \gamma \left(\frac{1}{2} \mathbf{E}^2 - E_y^2 \right) - \partial_x \gamma E_x E_y \right] \tag{3.53}$$

If we compare these two last expressions to the virtual work expression (3.45 3.43) we immediately observe the similarity.

Equivalence

If we expand the two gradients of γ by using the definition of the barycentric coordinates functions and their gradients 2.19 2.20 2.21 we have:

$$\partial_x \gamma = \frac{1}{2A_K} \left(r_1 (a_2^2 - a_2^3) + r_2 (a_2^3 - a_2^1) + r_3 (a_2^1 - a_2^2) \right) \tag{3.54}$$

$$\partial_y \gamma = \frac{1}{2A_K} \left(r_1 (a_1^3 - a_1^2) + r_2 (a_1^1 - a_1^3) + r_3 (a_1^2 - a_1^1) \right) \tag{3.55}$$

for completing the proof, we need only to show the equvalence between $\partial_x \gamma$, $\partial_y \gamma$ and the equvalents factor in the virtual work expressions 3.45, 3.43:

Then we must proof these two equivalences:

$$A_K \partial_x \gamma = (a_x d - c_x b) = -(-b_y d + d_y b) \tag{3.56}$$

$$A_K \partial_y \gamma = (a d_y - c b_y) = -(c a_x - a c_x) \tag{3.57}$$

Now we develop these four expressions by using the definitions 3.28, 3.30 and 3.31; then, we finally proof the equivalence between the two methods:

At first we expand the expression in 3.56

$$\begin{aligned} (a_x d - c_x b) &= (r_2 - r_1)(a_2^3 - a_2^1) - (r_3 - r_1)(a_2^2 - a_2^1) && (3.58) \\ &= r_1 (a_2^2 - a_2^3) + r_2 (a_2^3 - a_2^1) + r_3 (a_2^1 - a_2^2) && (3.59) \end{aligned}$$

$$
\begin{aligned}
(-b_y d + d_y b) &= -(r_2 - r_1)(a_2^3 - a_2^1) + (r_3 - r_1)(a_2^2 - a_2^1) && (3.60)\\
&= r_1(a_2^3 - a_2^2) + r_2(a_2^1 - a_2^3) + r_3(a_2^2 - a_2^1) && (3.61)
\end{aligned}
$$

we expand also the expression in 3.56

$$
\begin{aligned}
(ad_y - cb_y) &= -(r_2 - r_1)(a_1^3 - a_1^1) + (r_3 - r_1)(a_1^2 - a_1^1) && (3.62)\\
&= r_1(a_1^3 - a_1^2) + r_2(a_1^1 - a_1^3) + r_3(a_1^2 - a_1^1) && (3.63)
\end{aligned}
$$

$$
\begin{aligned}
(ca_x - ac_x) &= (r_2 - r_1)(a_1^3 - a_1^1) - (r_3 - r_1)(a_1^2 - a_1^1) && (3.64)\\
&= r_1(a_1^2 - a_1^3) + r_2(a_1^3 - a_1^1) + r_3(a_1^1 - a_1^2) && (3.65)
\end{aligned}
$$

Now the relations 3.56 and 3.57 are clearly equivalents. Then the eggshell and the virtual work methods are equivalents.

3.2.3. Conclusion

After these proof we conclude that the eggsell and the virtual work methods are equivalents if the shell is the same. We can also conclude that the maxweel stress tensor method is a particular case of the virrtual work method.
Then the only realistic comparison between these three methods is the velocity of the code and the implementaton.

4

Intepolations procedures

In this chapter we describe some procedure for generating the surrounding volume around the rigid body $\partial\Omega_1$. These interpolations values can be equivalently used as *surrounding shell* if we use the *Eggshell method* or as a *virtual distorted area* if we use the *virtual work method*. The main goal of this chapter and the next one (5) is to investigate the quality of the algorithms if we work with different interpolations procedures and problems configurations.

We can divide these procedures into two big classes:

- Mesh based interpolations: In this case the value assigned to the node is based on his position in the graph of the mesh grid.

- Geometric interpolations: The value is a function of the Cartesian coordinates of the node.

4.1. One on the boundary

This is the simplest and the most direct method. We simply assign 1 to the nodes that lie on the boundary $\partial\Omega_1$ and zero on all others nodes.
This method offers the advantage that it works with any shapes of $\partial\Omega_1$. It is , also, very simple to implement.
The big disadvantage is that it converges very slowly. So for obtaining a good result we must perform a big numbers of refinements.
As described in chapter 3 the result derived with this interpolation is equivalent to the result derived by *Maxwell Stress Tensor procedure* (2.1).

4.2. Boundary Layers

This is a *Mesh based interpolation* and can be considered as an extension of the *One on the boundary* interpolation.

The main idea is to set to 1 the values on the boudary. The values of the nodes in a sigle layer aroud the body $\partial\mathbf{\Omega_1}$ are constant.

The algorithm for building the interpolation is:

L := Number of Layers.
B := Get all nodes on the boundary $\partial\mathbf{\Omega_1}$
Set to 1 *the value of* r_i *on the nodes* $\in B$
For i := 1 ... L-1
    S := Get all nodes adjacent to B where the value of r_i *is* 0
    Set to $1 - i * \frac{1}{L}$ *the values of* r_i *on the nodes* $\in S$
    B := S
end For

The force computed with this method and the one on the boudary interpolation are asymptotically equivalent.

4.3. Harmonic Interpolation

In this case we use the solution of the *Laplace equation* for interpolating the values of the solution u_N as r_i on the nodes.

We set to 1 the Dirichlet boundary conditions on the rigid body $\partial\mathbf{\Omega_1}$ and to 0 on all others boundaries and we use the on the nodes.

With this method we obtain good resuls with and a good convergence rate. It is easy to implement. In a real application we must only change the boundary conditions in the original problem or, if it is possible, we can uses the solution of the original problem by simply rescaling the values.

4.3.1. Partial Harmonic

In this case we set to 1 the B.C. on the rigid body $\partial\mathbf{\Omega_1}$ and to $-a$ (where a is normally set to 1) on the others boundaries. For interpolating the values r_i on the nodes we use the values of the solution if they are positive and 0 otherwise.
This method guarantees a very good solutions and convergence rate. In ours tests it is the best.

4.4. Interpolation Around a generic shape

With this procedure we generate a generic interpolation around an arbitrary shape (the boundary $\partial\mathbf{\Omega_1}$). The value R_{Lim} is the size limit of the *shell* (If the distance between $\partial\mathbf{\Omega_1}$ and the node is greater than R_{Lim} r_i is set to 0).
The pseudo code of the procedure is:

N := Set of all nodes in the Meshes
For Each $n_i \in N$
    d *:= Minimal distance between* n_i *and* $\partial\mathbf{\Omega_1}$
    $v := f(d)$
    if $v > 0$ *then*
        $r_i := v$
    else
        $r_i := 0$
    end if
end For Each

Generic exponetial

If we desire an exponential interpolation around the boundary we use the following definition of $f(d)$:

The value $R_I > 0$ is an user defined parameter

$$f(d) := \frac{\exp\left((d + R_I)/R_I\right) - 1) - \exp\left((R_{Lim} + R_I)/R_I\right) - 1)}{1 - \exp\left((R_{Lim} + R_I)/R_I\right) - 1)} \tag{4.1}$$

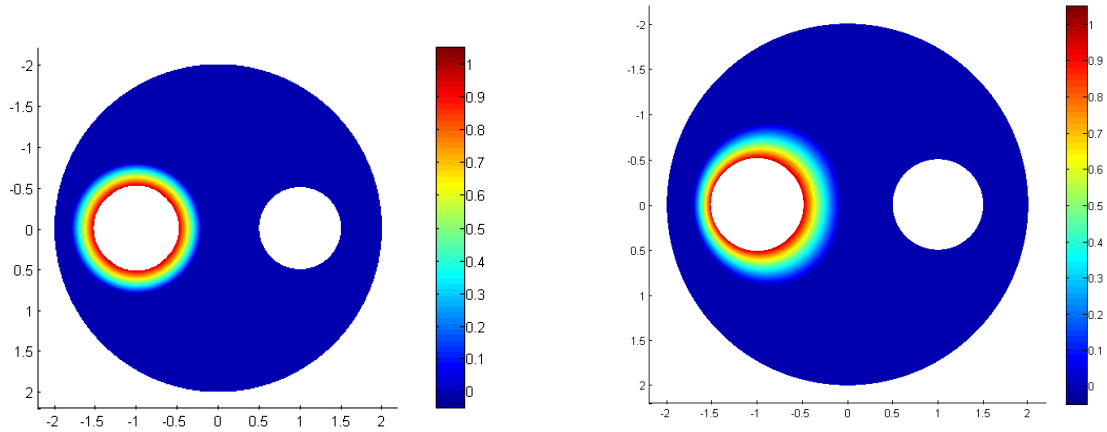
Generic linear

The linear interpolation can be simply implemented with this definition of $f(d)$:

$$f(d) := 1 - dR_{Lim}^{-1} \tag{4.2}$$

4.5. Interpolation around a circle

If $\partial\mathbf{\Omega_1}$ is a circle the values r_i can be directly interpolate with the procedures described above. d is the distance between the centre of $\partial\mathbf{\Omega_1}$ and the node.



(a) (b)

Figure 4.1.: Two examples of interpolations: (a) Generic Exponential. (b) Partial harmonic

The two following procedures can be implemented in a very efficient code and returns good results with a good convergence rate.

Generic Exponential

$$r_i := \frac{\exp\left(d/R_1 - 1\right) - \exp\left(R_2/R_1 - 1\right)}{1 - \exp\left(R_2/R_1 - 1\right)} \tag{4.3}$$

if $r_i < 0$ then $r_i := 0$.

where R_1 is the radius of $\partial\mathbf{\Omega_1}$ and R_2 is the radius of the *shell* and d is the distance between the center of $\partial\mathbf{\Omega_1}$ and the node.

Clearly we consider only the positives values.

Linear *Conic Interpolation*

In this case we interpolate the values in the *shell* with a cone that is defined with two circles: The first one is $\partial\mathbf{\Omega_1}$ with radius R_1 and center $C_1 := [x_1; y_1]$ the second is R_2 and center $C_2 := [x_2; y_2]$ where $R_2 > R_1$.

At first we define the parametric equation of the cone:

$$x(t,\theta) = (R_2 + t * (R_1 - R_2)) * cos(\theta) + (x_1 - x_2) * t \tag{4.4}$$
$$y(t,\theta) = (R_2 + t * (R_1 - R_2)) * sin(\theta) + (y_1 - y_2) * t \tag{4.5}$$
$$z(t,\theta) = t \tag{4.6}$$

The apex of the cone is found where t has the value of $t_v = \dfrac{-R_2}{R_1 - R_2}$.
Now the problem is, find the value of the parameter t on the node $[N_x; N_y]$ coordinates.

The value r_i on each node is $r_i = \dfrac{t-1}{t} * t_v$.

4.6. Coarsed displacement Interpolation

This procedure is based on the refinement process. The main idea is to perform an interpolation on a mesh structure after a little number on refinement (normally two or three). After this process we continue the refinement, on the modified mesh, up to the desired level:
The procedure is:

Given the *basic mesh structure* and the *refined structure*. 1) Perform some refinement on the basic mesh data structure.
2) Move the nodes along an arbitrary direction according with some interpolation rules.
3) Continue the refinement process up to the desired level.
4) Use the difference between the coordinates in the *refined structure* and the news coordinates of the node for generating the interpolations vales.

In the point 2 we can use a harmonic interpolation or any others rules described in this chapter. The big disadvantage of this method is that it requires a second mesh data structure and it is slow. It returns good results and it has a good convergence rate.

5

Test And Results

5.1. Test procedure

The main goal of the tests is to study the evolution of the error as a function of the DOF (degree of freedom) with different interpolations procedures.

As first we define the relative error:

$$Relative\ Error = \frac{|\mathbf{F}_n| - |\mathbf{F}_{best}|}{|\mathbf{F}_{best}|} \tag{5.1}$$

Where $\mathbf{F}_n$ is the force estimated after n refinements and the $\mathbf{F}_{best}$ is the force estimated with the analytical solution if it is known, or a reference force estimated after a big number of refinements iterations.

Another interesting parameter is the evolution of the slope of the error curve and the average slope. With the slope we measure the *convergence rate* of the tested procedures.

If the analytical solution u is known the force can be calculated by integrating analytically the Maxwell stress tensor over the inner surface $\partial\mathbf{\Omega_1}$.

The degree of freedom (DOF) parameter of a mesh structure is defined by

$$DOF := Number\ Of\ Edges + Number\ Of\ Nodes \tag{5.2}$$

In the followings tests we never report the *egg shell method*, but as proofed in the chapter 3 the virtual work procedure and Egg Shell procedure are equivalents so all the conclusions about the

virtual work are the same that we could have with the eggshell method. We have observed only a very little (ca. 10^{-15}) difference between the two methods, due to the numerical approximation.

5.2. Virtual Work results compared with an analytical result

The first, and the most obvious, test is to compare an analytical result with the numerical result. With this kind of test we can do a reliable comparison between the different interpolations methods and decide which is the better.

5.2.1. Problem

The problem is a Dirichlet boundary value problem: The domain Ω is defined by two concentric circles $\partial\mathbf{\Omega_1}$ and $\partial\mathbf{\Omega_2}$. The radius of $\partial\mathbf{\Omega_1}$ is 1 and of $\partial\mathbf{\Omega_2}$ is 2.

$$\nabla^2 u = 0 \tag{5.3}$$

$$u\mid_{\partial\mathbf{\Omega_1}} = 1 + \cos\theta + cos^3\theta \quad on\ \partial\mathbf{\Omega_1} \tag{5.4}$$

$$u\mid_{\partial\mathbf{\Omega_2}} = 0 \quad on\ \partial\mathbf{\Omega_2} \tag{5.5}$$

With the solution:

$$u = A_0 + B_0 \log r + rA_1 \cos\theta + \frac{1}{r}C_1 \cos\theta + r^3 A_3 \cos 3\theta + \frac{1}{r^3}C_3 cos3\theta \tag{5.6}$$

Where

$$A_0 = \frac{a_0 \log R_2}{2(\log R_2 - \log R_1)} \tag{5.7}$$

$$B_1 = \frac{a_0 \log R_2}{2(\log R_2 - \log R_1)} \tag{5.8}$$

$$A_1 = \frac{-R_1 a_1}{R_2^2 - R_1^2} \tag{5.9}$$

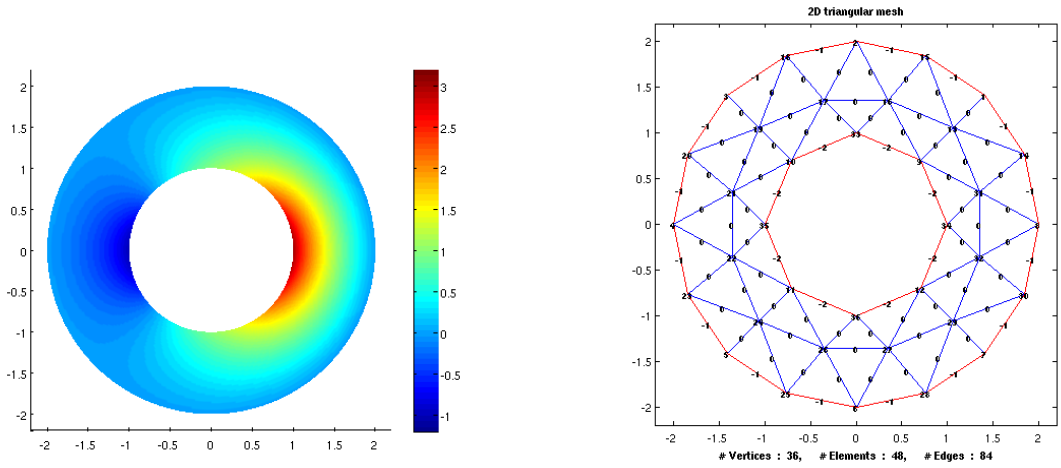
$$C_1 = \frac{(R_1 R_2)(R_2 a_1)}{{R_2}^2 - R_1^2} \tag{5.10}$$

$$A_3 = \frac{-R_1^3 a_3}{R_2^6 - R_1^6} \tag{5.11}$$

$$C_3 = \frac{(R_1 R_2)^3 (R_2^3 a_3)}{R_2^6 - R_1^6} \tag{5.12}$$

where $a_0 = 2$, $a_1 = \frac{7}{4}$ and $a_3 = \frac{1}{4}$ are the *fourier coefficients* of $u\mid_{\partial\mathbf{\Omega_1}}$.

The analytical solution of the force has been calculated by performing the surface integral of the *Maxwell stress tensor* (2.1) on $\partial\mathbf{\Omega_1}$. For obtaining this result we have used the well known symbolic manipulation software `Maple`.



(a) (b)

Figure 5.1.: Solution (a) and mesh structure (b) of the problem

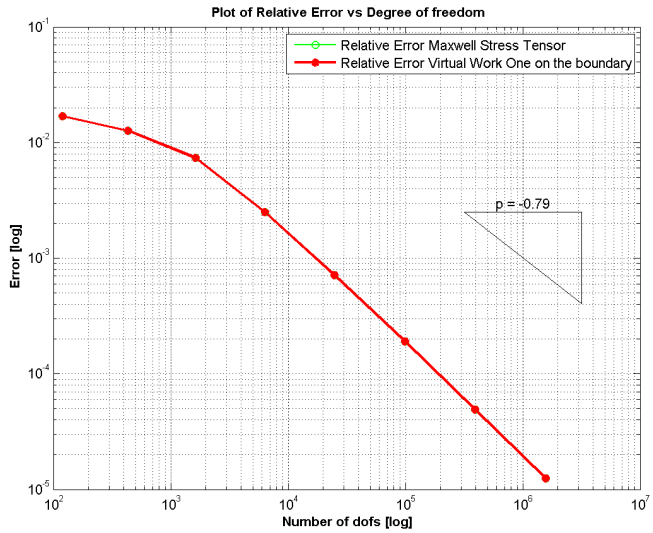
Settings

We have repeated the same test with the different interpolations. For each test we report two charts.

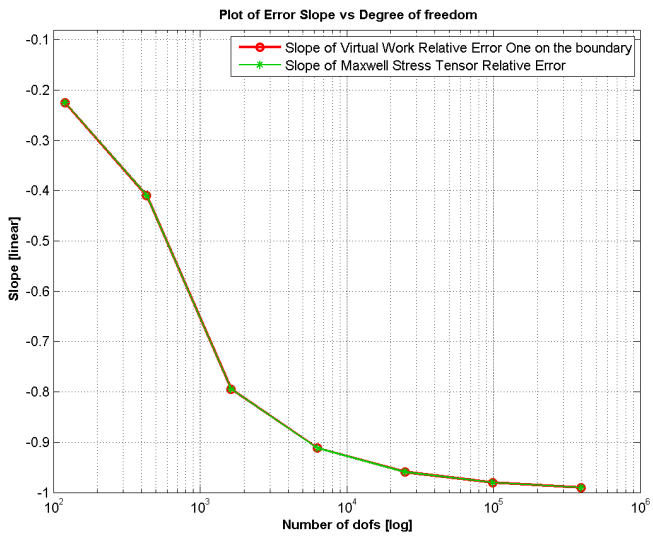
In the first chart we report the evolution of the *relative error* as a function of the DOF, in the second chart we report the evolution of the slope. We use always a log-log scale.

With these two measures, we can examine the the behaviour of the algorithms.

5.2.2. Result for One on the boudary



(a)

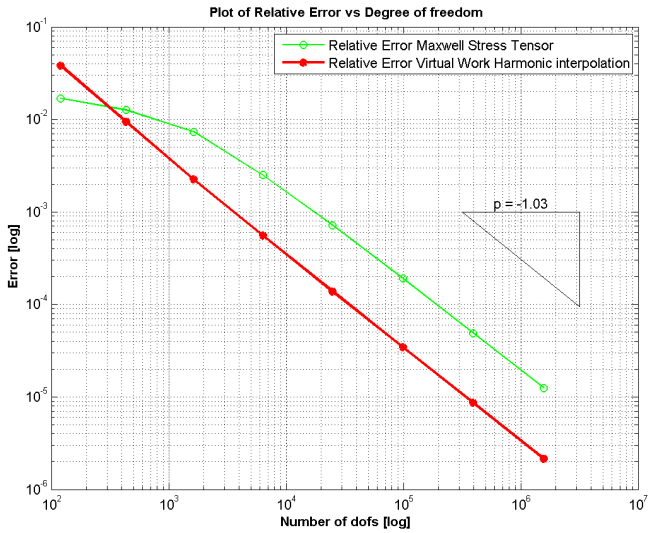


(b)

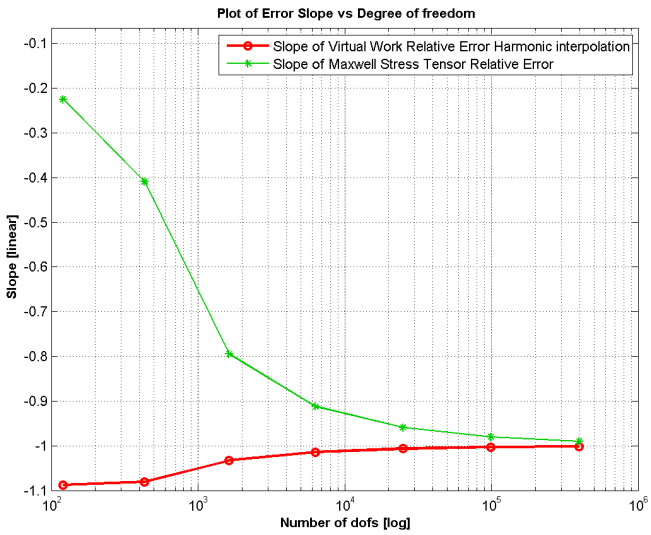
Figure 5.2.: Comparison between Virtua Work (with 1 on the boudary) and Maxell Stress Tensor (a) Relative Error. (b) Slopes

In the direct comparison between the *maxwell stress tensor* and *one on the boudary* interpolation the two results are clearly identical (see also chapter 3).

5.2.3. Results for Harmonic interpolation



(a)



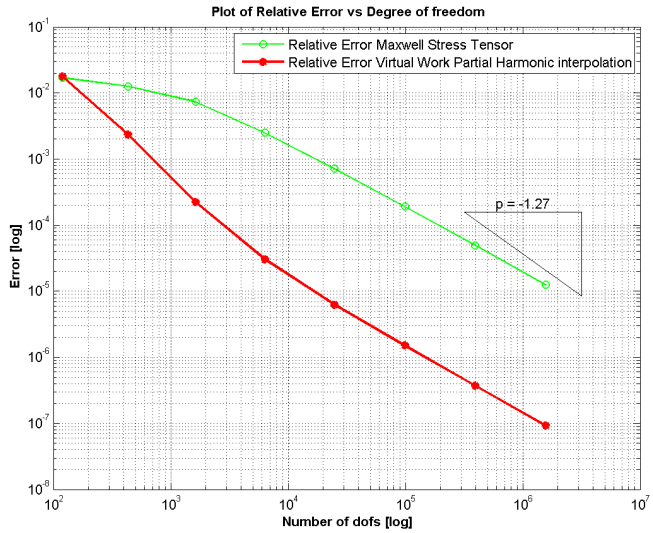
(b)

Figure 5.3.: Comparison between Virtua Work (harmonic interpolation) and Maxell Stress Tensor

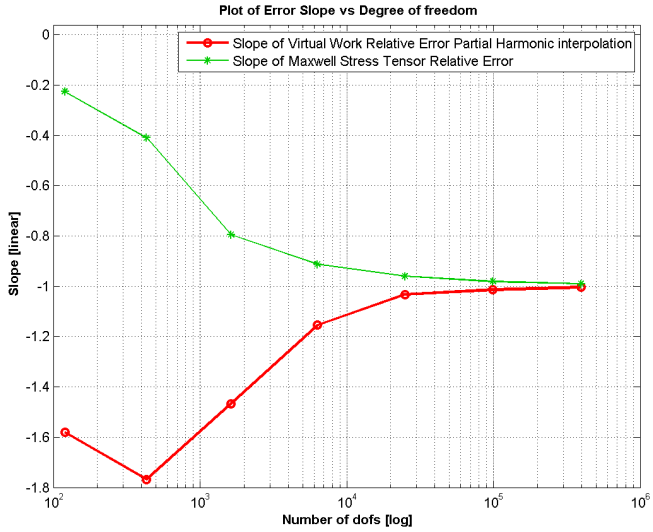
(a) Relative Error (b) Slopes

The harmonic interpolation guarantee a good and stable convergence rate.

5.2.4. Results for Partial Harmonic interpolation



(a)

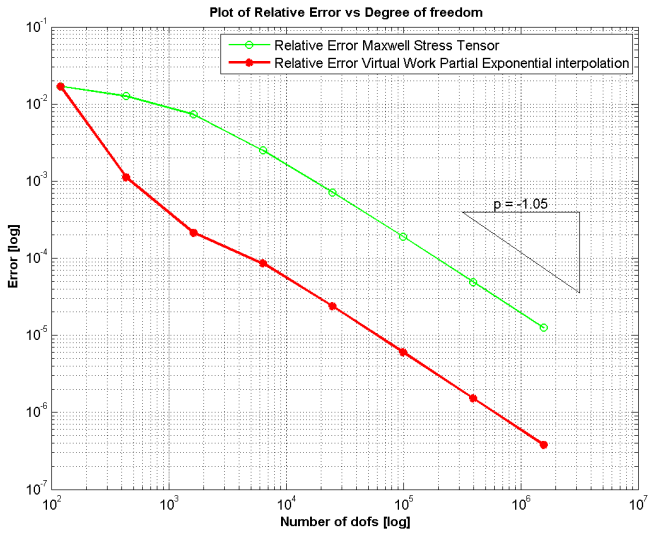


(b)

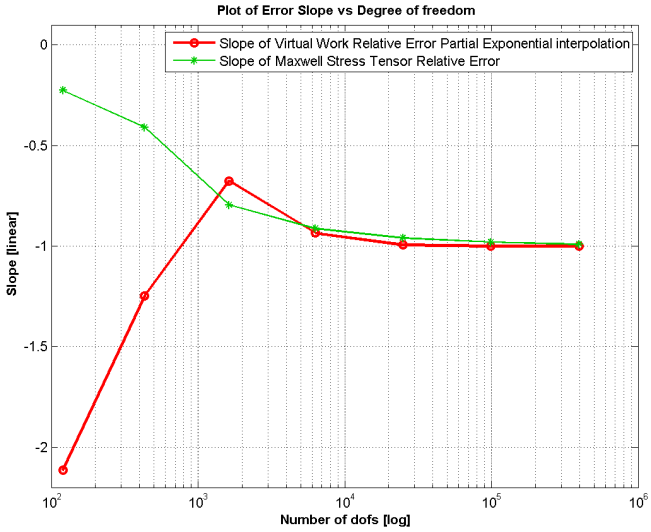
Figure 5.4.: Comparison between Virtua Work (partial harmonic interpolation) and Maxell Stress Tensor
(a) Relative Error (b) Slopes

In this test we have used 1 on the virtual moved boundary and -1 on the other boundary (see also 4.3). The partial harmonic interpolation procedure has produced the best convergence rate in our tests.

5.2.5. Results for Partial Exponential interpolation



(a)

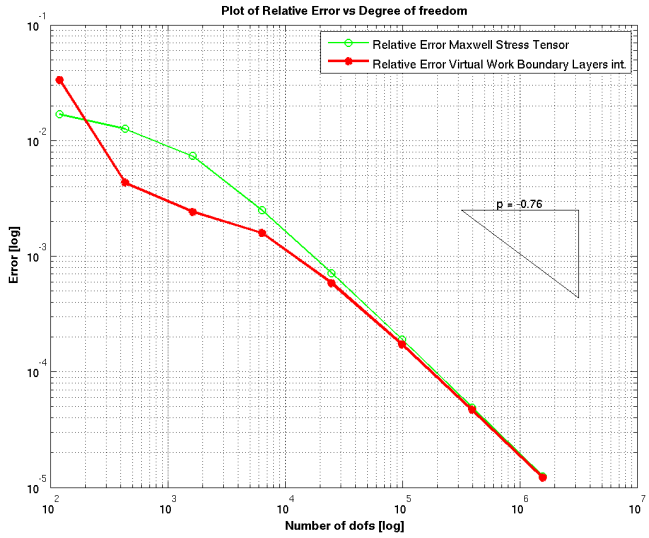


(b)

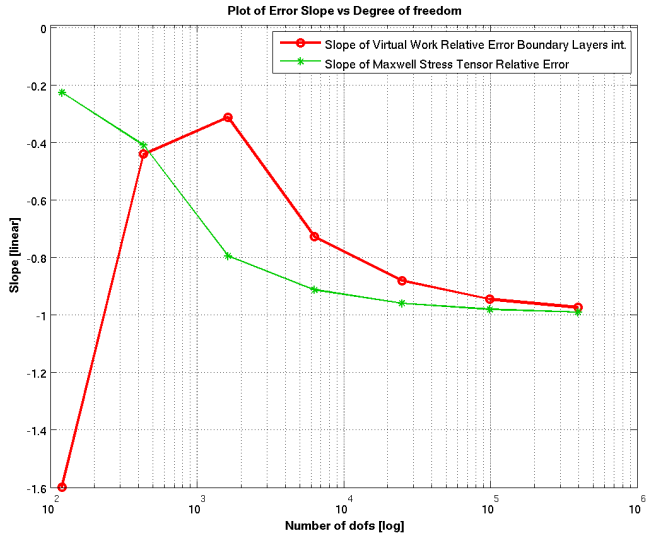
Figure 5.5.: Comparison between Virtua Work (partial exponential interpolation) and Maxell Stress Tensor
(a) Relative Error (b) Slopes

In this test we have used 1.3 as external radius.

5.2.6. Results for Boudary Layers interpolation



(a)



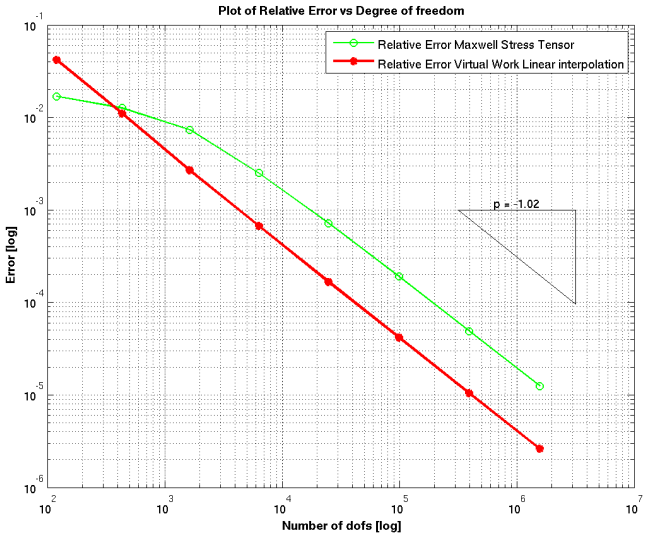
(b)

Figure 5.6.: Comparison between Virtua Work (boudary layers interpolation) and Maxell Stress Tensor

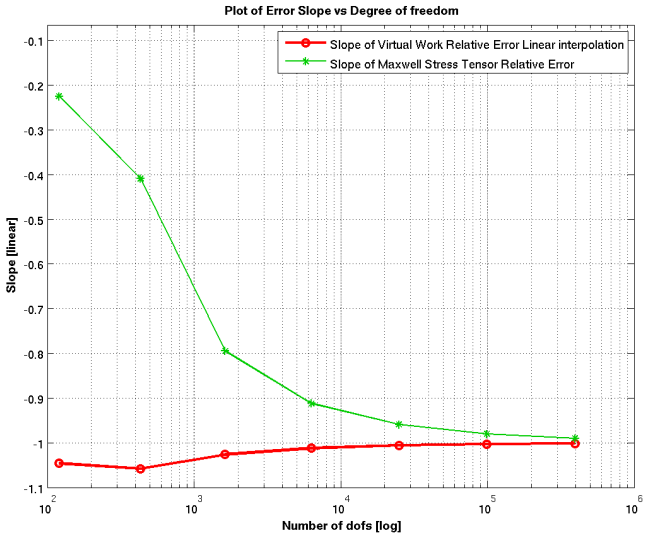
(a) Relative Error (b) Slopes

In these charts we observe that the force obtained with the *Maxwell stress tensor* and the *boudary layers* interpolations, asymptotically converge to the same solution.

5.2.7. Results for Linear interpolation



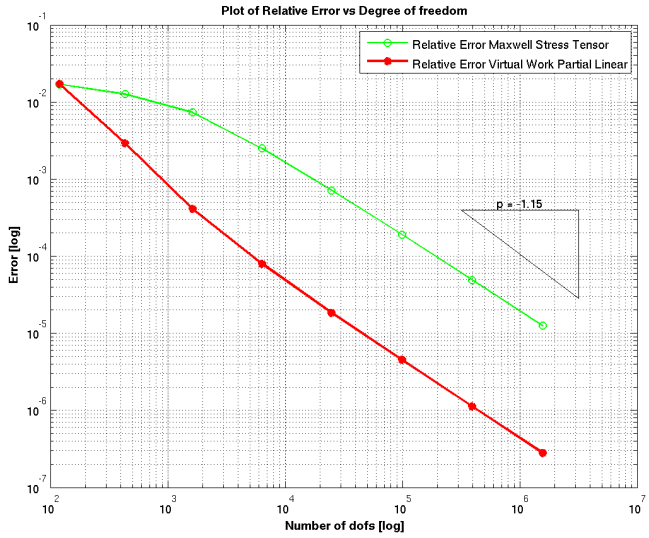
(a)



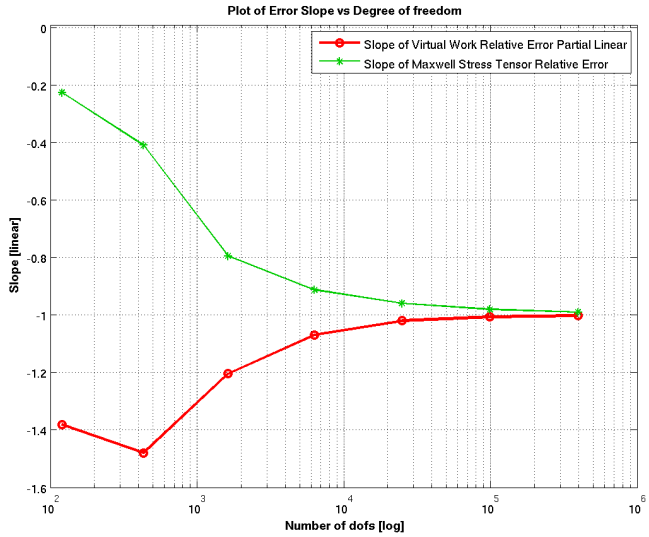
(b)

Figure 5.7.: Comparison between Virtua Work (linear interpolation) and Maxell Stress Tensor (a) Relative Error (b) Slopes

5.2.8. Results for Partial Linear interpolation



(a)



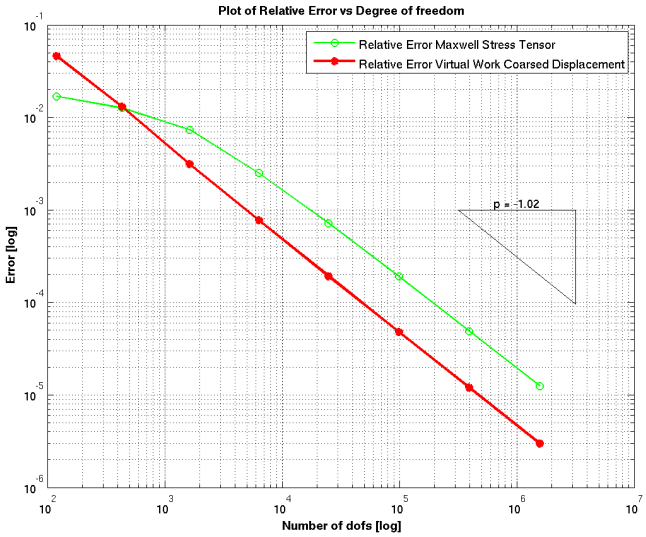
(b)

Figure 5.8.: Comparison between Virtua Work (partial linear interpolation) and Maxell Stress Tensor

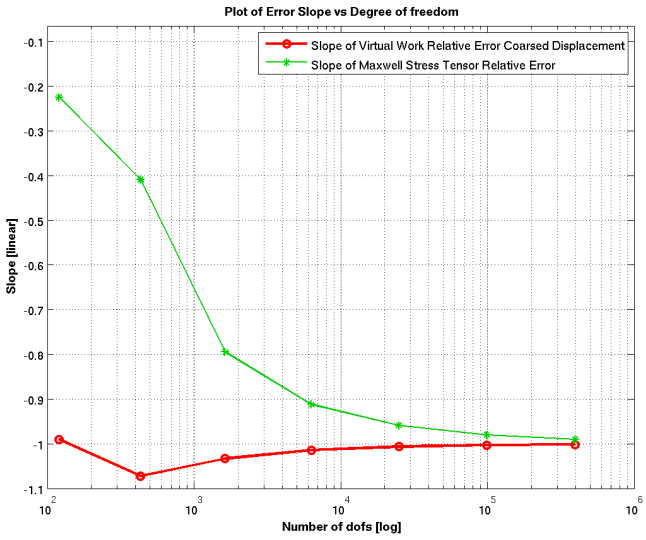
(a) Relative Error (b) Slopes

In this experiment the radius of the external interpolation circle is $0.9 * R_2$ (see laso 4.5).

5.2.9. Results for Coarsed Virtual displacement interpolation



(a)



(b)

Figure 5.9.: Comparison between Virtua Work (coarsed displacement interpolation) and Maxell Stress Tensor
(a) Relative Error (b) Slopes

5.2.10. Commets and observation

By observing the results, we can conclude that the best convergence rate has been obtained with the *partial harmonic interpolation*.

If we examine the charts of the slopes we observe that the slope of all method asymptotically

converge to -1 in any case. So the procedures, from this point of view, become equivalent with a big number of DOF. An intuitive explanation of this phenomenon is that with a very big number of DOF the numerical integration of the Maxell stress tensor converges to the analytical solution. Thus the convergence rate improves with the number of DOF.

All methods based on a geometric interpolation (4) normally have an average slope of ~ 1. They have a very good convergence rate with a little numbers DOF that decrease with the growing of the DOF. Instead the *Mesh based interpolations* starts with a low convergence rate that grows with the number DOF. But in any case the result obtained with the geometric are better.

5.3. Virtual Work results compared with the best result

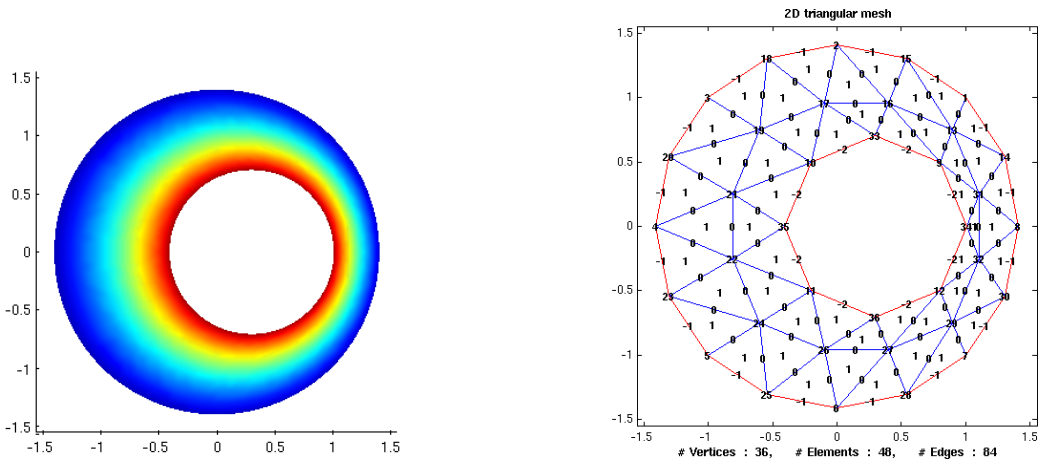
5.3.1. Tests with the translation of the inner object

For performing the tests, we have translated the inner rigid body $\partial\mathbf{\Omega_1}$ of 0.05, 0.1, 0.2 and 0.3 along the x axis and we have estimated the force on each configuration by refining (h-refinement) the mesh from 1 to 8 times.

We have repeated the full test with two concentric circles.

In the resulting charts we report the errors for each translation of the object.

In the figure 5.3.1 we clearly observe the distortion of the mesh structure after the translation.



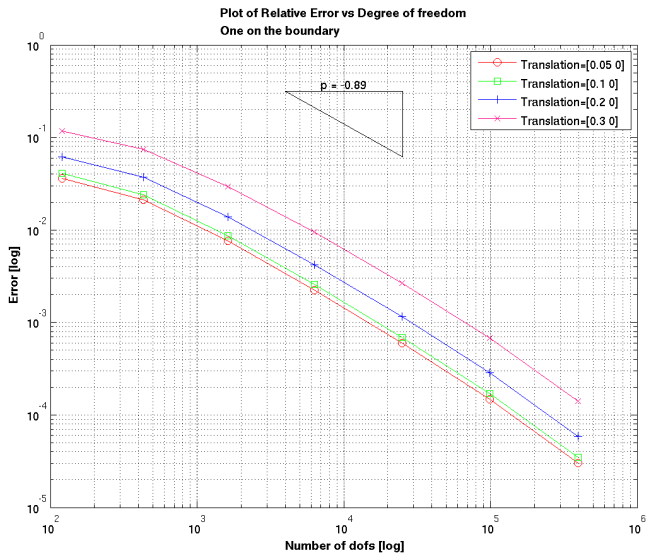
(a) (b)

Figure 5.10.: Example with the inner object diplaced of 0.3. (a)Solution. (b) Mesh

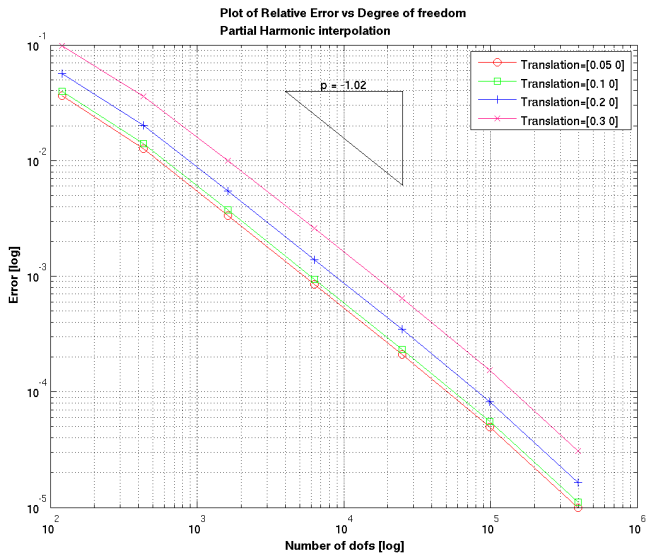
With this sets of tests we can observe the evolution of the error as a function of the mesh distortion. In the charts we report the evolution of the error of each translation.

We do not know the analytical solution of this problem. So we perform $n + 1$ refinement and we use the resulting force $\mathbf{F}_{best}$ obtained with $n + 1$ refinement as a reference value. Thus the slopes and convergence rate are different respect to the test in the preview section (5.2).

5.3.2. Results

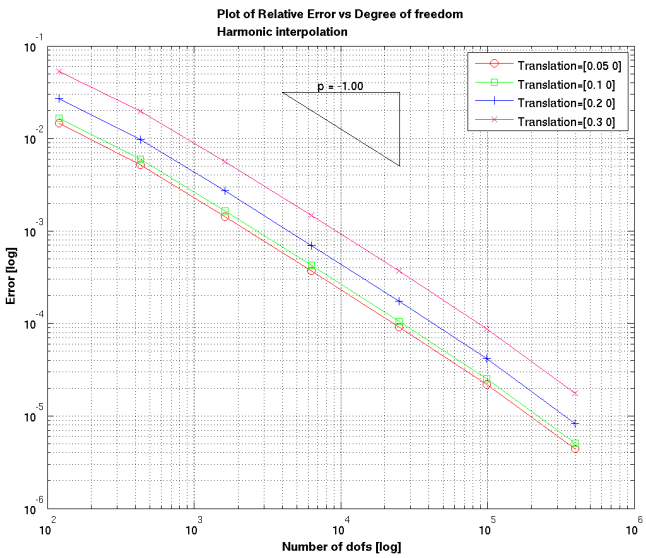


(a)

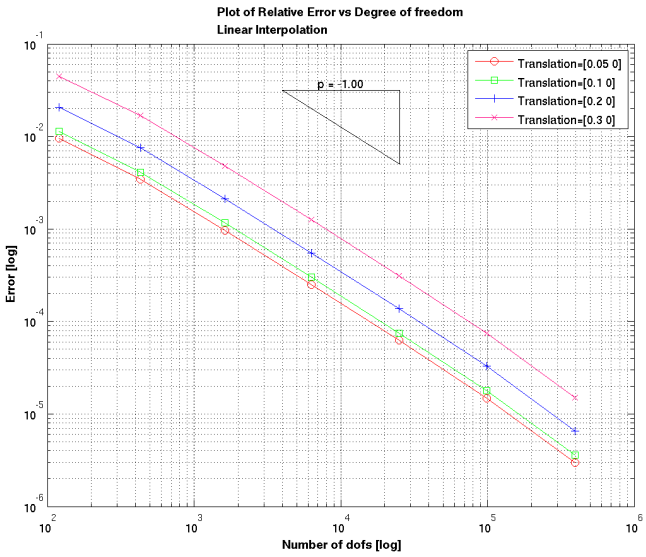


(b)

Figure 5.11.: (a) One on the bondary (b) Partial Harmonic

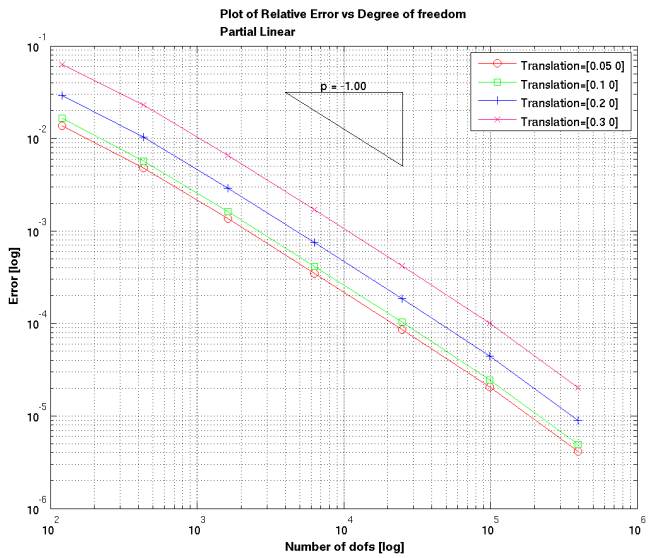


(a)

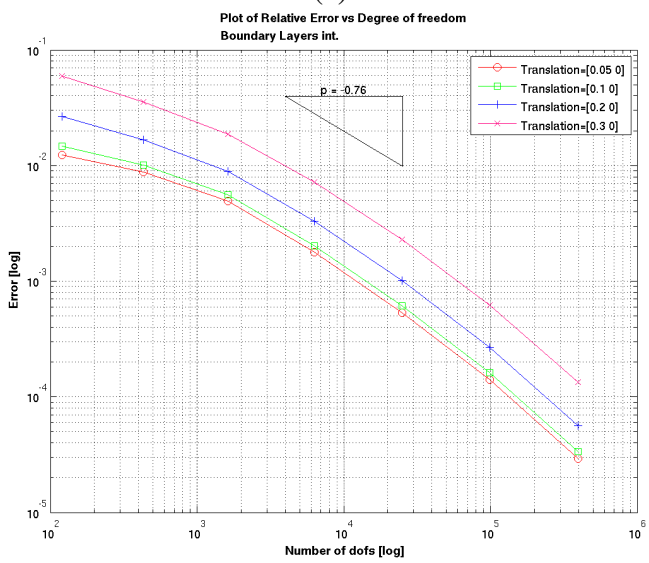


(b)

Figure 5.12.: (a) Harmonic (b) Linear



(a)



(b)

Figure 5.13.: (a) Partial Linear (b) Boubdary layers

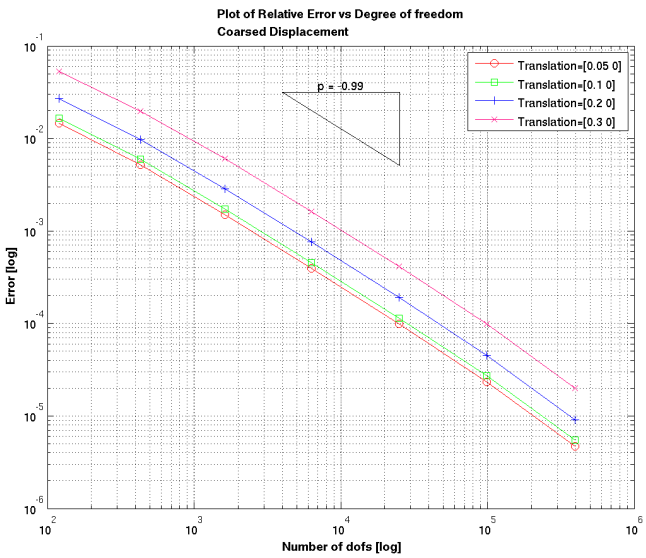


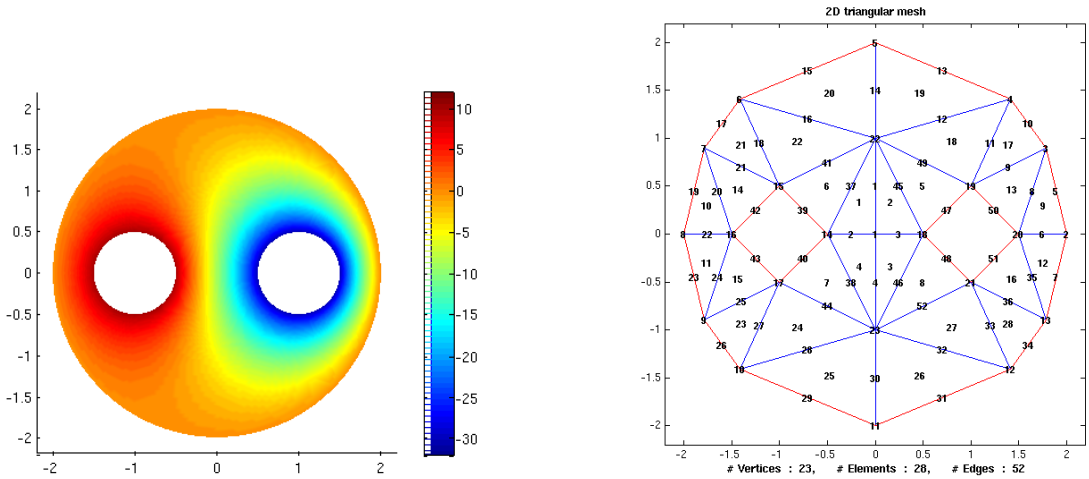
Figure 5.14.: Coarsed interpolation

5.3.3. Conclusion

In these tests, we observe that the mesh structure influence the result in any case. Thus the *virtual work* and the *Maxwell stress tensor* procedures are influenced by the mesh structure. We also observe that with a logarithmic scale the difference between the different errors curve stay constant, this effect is due to the regularity of the mesh and distortion.

5.3.4. Tests with the double circle configuration

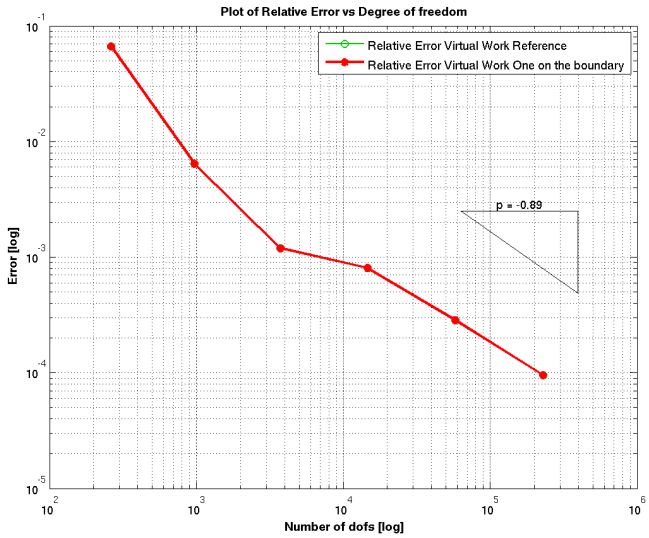
In this section we have done some experiments with a configuration with a double inner circle. In this case as a reference result we use, in the firsts charts, the force computetd with a partial harmonic intepolation. In the second chart we use the force computed after n+1 refinements steps.



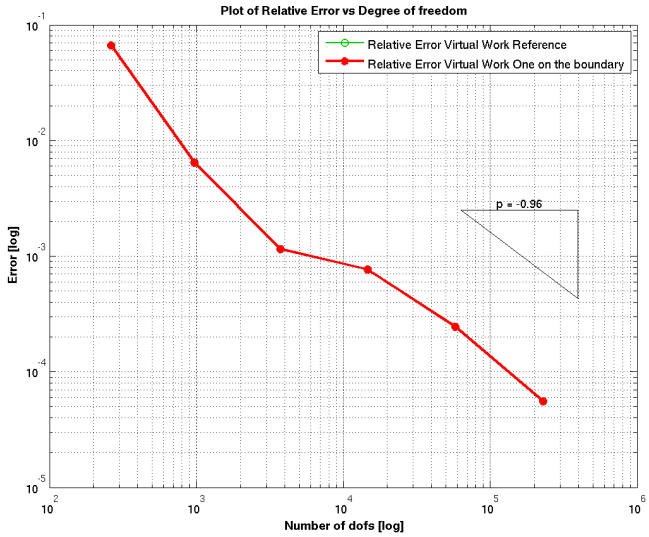
(a) (b)

Figure 5.15.: Example of a solution and the mesh data strucyure of the double inner circle (a)Solution. (b) Mesh

Results for One on the boudary



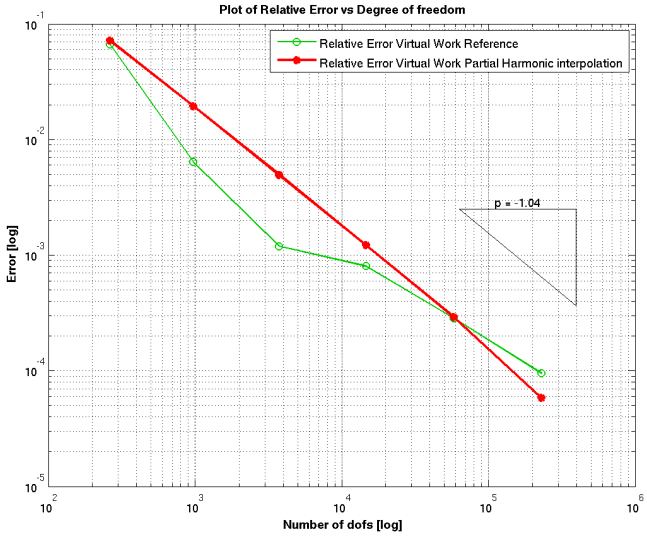
(a)



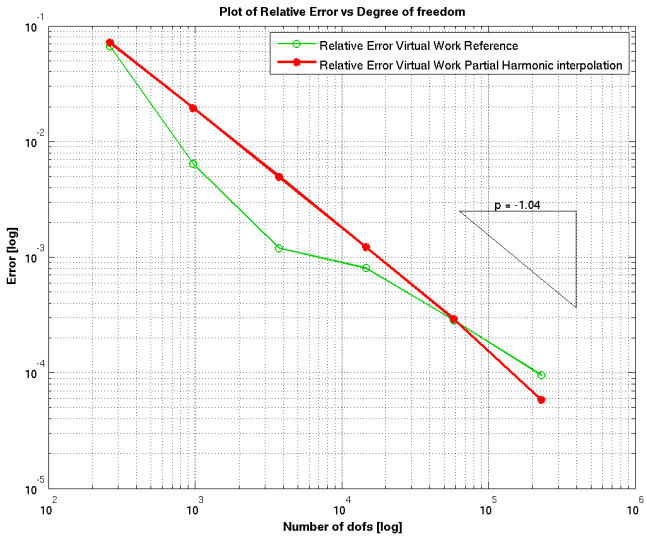
(b)

Figure 5.16.: Results of the double circle configuration conpared with the partial harmonic and itself (a) Partial hamonic (b) itself

Results for Partial Harmonic



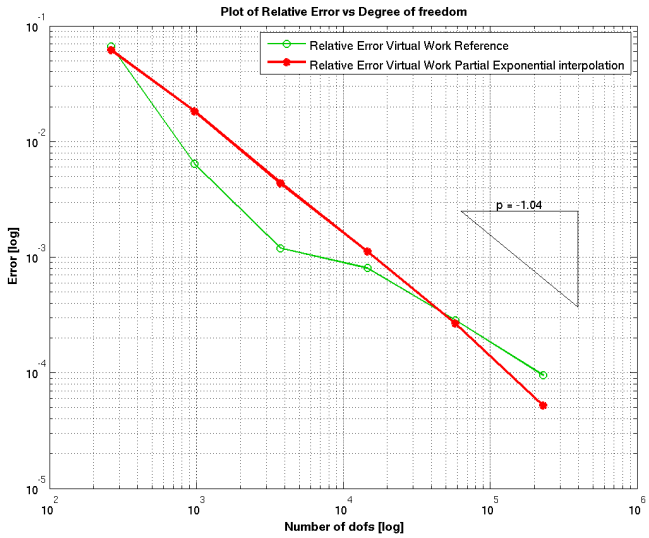
(a)



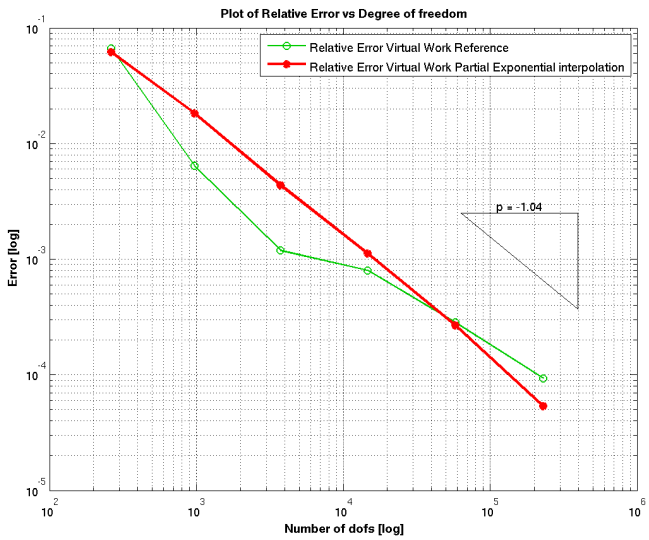
(b)

Figure 5.17.: Results of the double circle configuration conpared with the partial harmonic and itself (a) Partial hamonic (b) itself

Results for partial exponential



(a)



(b)

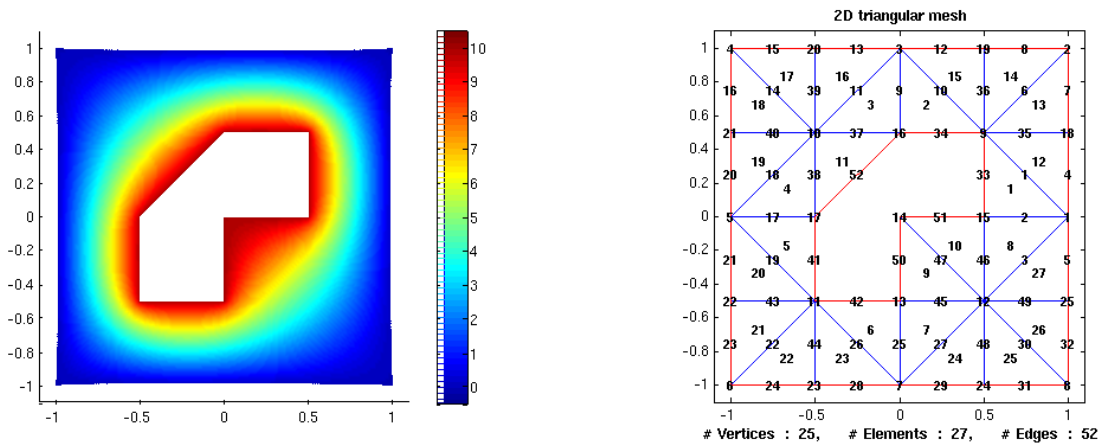
Figure 5.18.: Results of the double circle configuration conpared with the partial harmonic and itself (a) Partial hamonic (b) itself

Conclusion

We observe a very little difference between the two comparison methods. The interpolations procedures produce a similar results with respect to the results obtained in the preview section.

5.4. Tests with the *Aposteriori adaptive* refinement

In this section we study the effect of an adaptive and irregular refinement on the results. The irregularity of the mesh introduce a pseudo random behaviour in the convergence rate, but on the other hand, the adaptivity perform the refinement where the *discretization error* is big. Also in this case we do not know the analytical solution so we use a reference result.



(a) (b)

Figure 5.19.: Example of a solution and the mesh data strucyure of the LM data structure (a)Solution. (b) Mesh

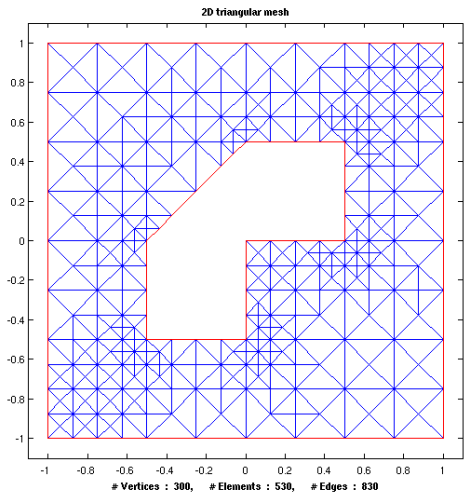
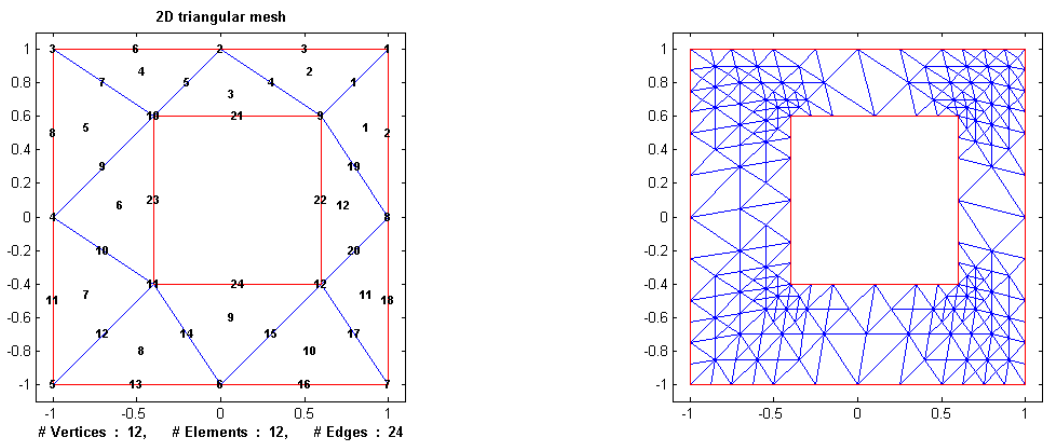


Figure 5.20.: The effect of the *A Posteriori* adaptive refinement on the 'LM' structure

The advantage of the adaptivity is that it refines the mesh only in the problematic region of Ω. Normally around the angle or where the magnitude of the gradient of u is big. We can obtain a good *discretization error* with a little number of DOF.

The goal of the experiment is to understand if the advantage of the adaptivity, counterbalanced



(a) (b)

Figure 5.21.: The original mesh(a) and the effect of the adaptivity (b) of the SQ data structure

with the disadvantages introduced by the irregularity of mesh, produce some benefit in the force computation.

Method

For performing these tests we use two meshes structure: the first one (named LM) is a complex concave object $\partial\Omega_1$ with a constant Dirichlet boundary condition.
The second configuration (named SQ) consists of two off centred squares.

The *reference force* in both cases has been obtained with the *partial harmonic interpolation* after 8 regular refinements.

In the followings charts we report the direct comparison between the error of the regular refinement and the error of the adaptive refinement, with the same interpolation method.

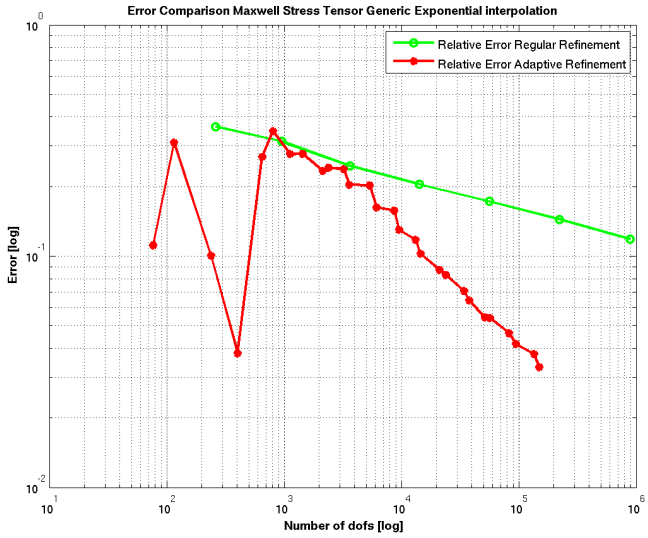
5.4.1. Effect of the angles in the computation

With the 'LM' test we have a non smooth (with angles) object in the computation. The first analytical effect is that the (unknown) solution u is described by an infinite Fourier series. As consequence, in the numerical solution, we have a big approximation error around the angles. This error directly influence the computation of the force with the *Maxwell Stress Tensor* with a very slow convergence rate. This effect is attenuated by the use of a large shell (or virtually distorted area) that involve also the regions where the 'high frequency' components of the solution u have a low influence.

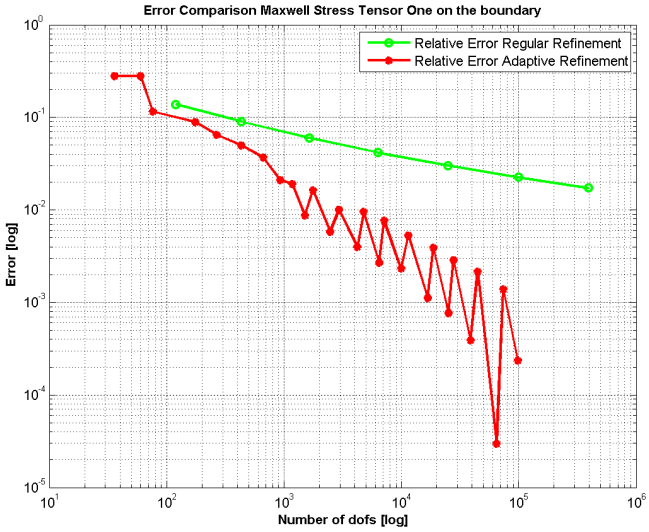
[h]	Force 5 ref.	Force 6 ref.	Force 7 ref.
Maxwell Stress Tensor	18.649	18.197	17.792
Partial Harmonic	16.050	15.984	15.935
Harmonic	15.813	15.863	15.873
Exponential	15.785	15.823	15.8438

In the table 5.4.1 we observe the effect (on the LM mesh) described above, where the convergence rate of the *Maxwell Stress Tensor procedure* is very slow and the error is big.

5.4.2. One on the boundary



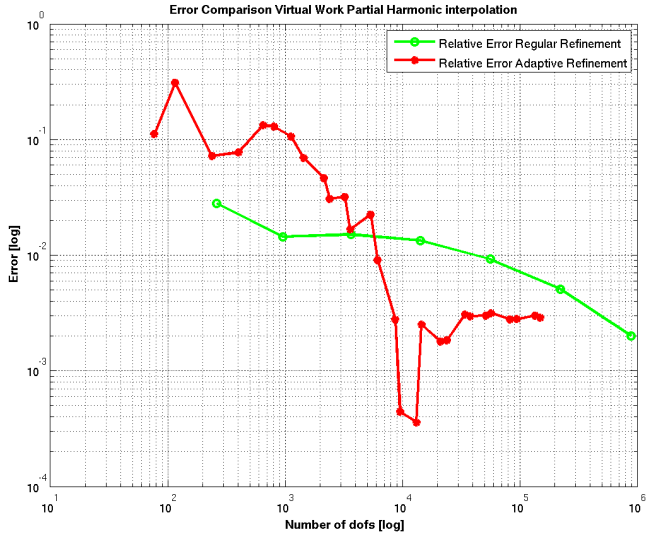
(a)



(b)

Figure 5.22.: Comparison between Adaptive and regular refinement with the *maxwell stress tensor* on the LM (a) and SQ (b) structures

5.4.3. Partial Harmonic

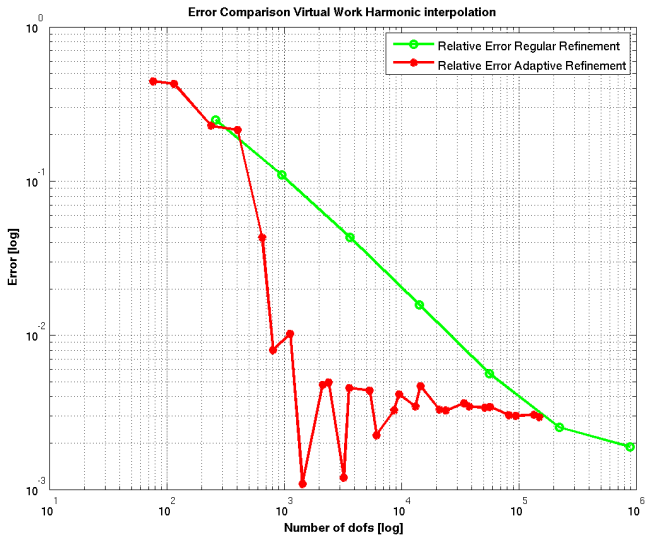


(a)

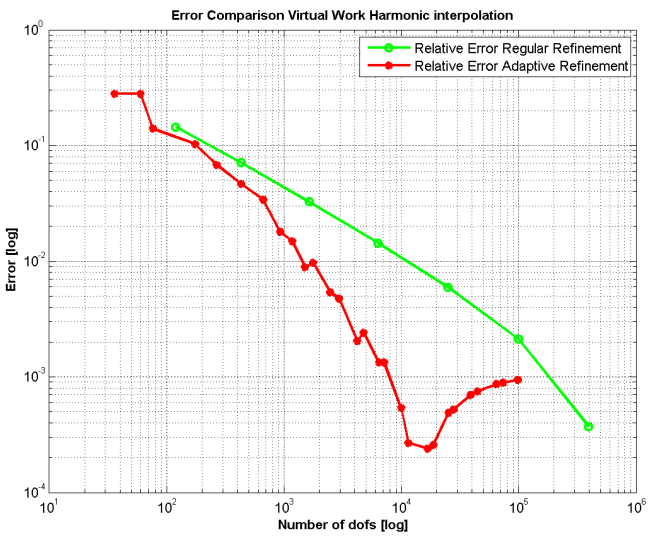
(b)

Figure 5.23.: Comparison between Adaptive and regular refinement with the partial harmonic interpolation on the LM (a) and SQ (b) structures

5.4.4. Harmonic



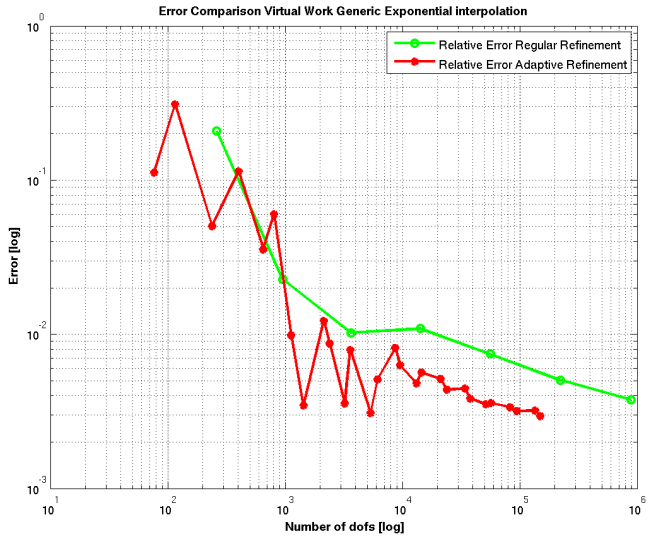
(a)



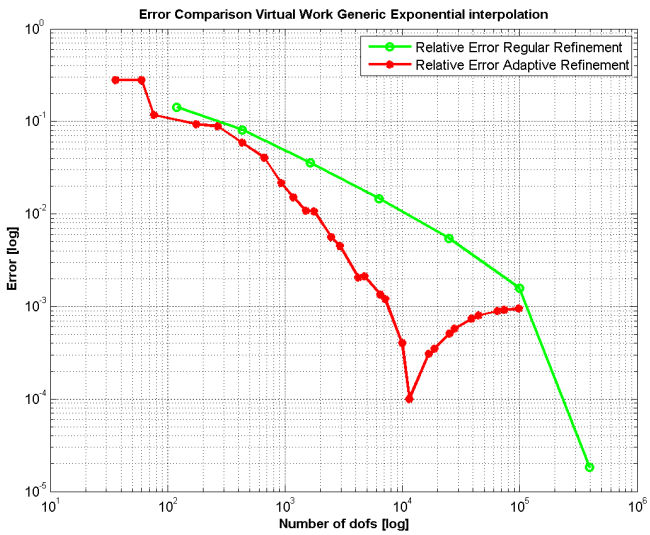
(b)

Figure 5.24.: Comparison between Adaptive and regular refinement with the harmonic interpolation on the LM (a) and SQ (b) structures

5.4.5. Generic Exponential



(a)



(b)

Figure 5.25.: Comparison between Adaptive and regular refinement with the harmonic interpolation on the LM (a) and SQ (b) structures

5.4.6. Conclusions and observations

If we observe the comparison we see that the adaptive refinement normally returns better results than the results obtained with a regular refinement. We also observe that the error has a big fluctuation in the first 3 or 4 refinements and only after some steps the convergence rate is about stabilized.

In the SQ structure we also observe a fluctuation of the error with a big number of DOF, and also an apparent lost of precision.

The big disadvantage of the *a posteriori* refinement is the slowness of the computation. Practically for computing the force with n refinements levels we must solve the full FEM solution of all previews levels.

6

Conclusion and Outlook

The first, and the most important, result of this work is the equivalence between the three methods. So the only comparison meter between these procedures is the efficiency and the simplicity of the code. From this point of view the better choice is the eggshell method. In our Matlab implementation the eggshell code is normally faster than the virtual work code and it is also very simple to implement. So the better choose in a linear finite element context is the eggshell algorithm.

In the study of the interpolation we have concluded that the interpolation procedure in any case is the partial harmonic interpolation and in general the procedures that return the better results are the ones that generate a sufficiently large shell that involve only a limited area around the rigid object.

We have also seen that the force estimated after an adaptive refinement is normally better than the result obtained after a regular refinement with the same number of DOF.

Porting the eggshell and the virtual work procedure in a 3D finite element context it is very easy so all the procedure are good methods for computing the force in a real world application.

A

Matlab functions

In the above chapter we describe the main functions used in the Matlab implementation of the virtual work, eggshell and Maxwell stress tensor procedures:

Given a Mesh with M nodes, N elements and V virtually distorted elements (or the elements included in the shell):

A.1. Force Computation

Virtual Work

```
function Fs = eval_Force_VirtualWork(Mesh,U,epsilon)
```

This function computes the force acting on the body with the virtual work procedure:

the arguments are:
Mesh: The mesh data structure.
U: M by 1 column vector that store the solution of the FEM problem.
epsilon: permittivity of the medium.

The data structure MESH should at least contain the following fields:
Mesh.VirtualEl: V by 1 vector specifying the indices of all virtually distorted elements.
Mesh.DispDerivative: M by 4 or M by 1 (sparse) matrix storing the displacements derivatives of each node.
If we use an M by 4 the first column is the derivative of x_i versus x, the second x_i versus y,,the

third y_i versus x and the fourth y_i versus y. The values of p_i must be assigned to the first and last column. With this code we should compute the force in any arbitrary direction if we known the derivatives. If we use an M by 1 simply assign the values of p_i on each node.
Mesh.Elements: N by 3 matrix specifying the elements of the mesh.
Mesh.Coordinates: M by 2 matrix specifying the vertices of the mesh.

Eggshell

```
function Fs = eval_Force_EggShell(Mesh,U,epsilon)
```

Compute the force with the eggshell procedure
the arguments are:
Mesh: The mesh data structure.
U: M by 1 column vector that store the solution of the FEM problem.
epsilon: permittivity of the medium.

The structure Mesh should at least contain the following fields:
Mesh.VirtualEl: V by 1 vector specifying the indices of all elements in the shell.
Mesh.DispDerivative: M by 1 (sparse) matrix storing the coefficients r_i for interpolating the shell.
Mesh.Elements: N by 3 matrix specifying the elements of the mesh.
Mesh.Coordinates: M by 2 matrix specifying the vertices of the mesh.

Maxwell stress tensor

```
function [F T] = eval_Force_MaxwellStressTensor(Mesh,U,epsilon,varargin)
```

Compute the force F and the torque T with the Maxwell stress tensor procedure.
the arguments are:
Mesh: The mesh data structure.
U: M by 1 column vector storing the solution of the FEM problem.
epsilon: permittivity of the medium.
varargin: should store an alternative center for the torque (the default center is [0 1]).

The strut MESH should at least contain the following fields:
Mesh.BdSurface: B by 2 matrix storing the x an y coordinates of the surface S in counter clockwise order. *Mesh.EdSurface*: B by 2 matrix storing all edges intersected by the surface.

```
function Mesh = build_bound_Surface(BdFlags,Mesh,varargin)
```

Build the surface S around the body marked with the boundary flag BdFlags.
Mesh: The mesh data structure.
BdFlags: Boundary flag of the body
varargin: should store an user defined center of the body.

For computing the force with the Maxwell stress tensor we must call the two previews functions with this sequence:

```
Mesh = build_bound_Surface(BdFlags,Mesh) ;
[F T] = eval_Force_MaxwellStressTensor(Mesh,U,epsilon) ;
```

A.2. Interpolation procedures

In this section we describe the functions for interpolating the virtually distorted area or the equivalent eggshell. These function add to the Mesh structure the field *DispDerivative* and *VirtualEl*.

Mesh.DispDerivative: is an M by 4 or an V by 1(sparse) matrix that store the virtual displacements derivatives of the nodes.

Mesh.VirtualEl: is an N by 1 vector that store the indices of the elements involved in the interpoletion (in the virtually distorted area or shell).

The *varargin*: optional arguments should store a character string made from one or all of the following elements:
s: Use a sparse matrix for allocating the field *Mesh.DispDerivative*.
o: Generate an M by 1 matrix in the *Mesh.DispDerivative*.
c: Plot the interpolation.

For using these functions the mesh data structure should requires at least the following field:
Mesh.Coordinates: The coordinates of the nodes.
Mesh.Edges: The edges of the mesh.
Mesh.BdFlags: The flags of the edges.
Mesh.Elements: The list of all elements.
Mesh.Edge2Elem: Associate the edges with the elements.

One on the boundary

```
function Mesh = virtual_Translation_BD(Mesh,BdFlags,varargin)
```

Gnerate the *one on the boundary* interpolation. Where the arguments are:
Mesh: The mesh data structure.
BdFlags: The flags of the virtually moved boundary.

varargin: The options described above.
It returns the updated Mesh data structure.

Interpolation with the solution

```
function Mesh = virtual_Translation_grad( BdFlags , U , Mesh , varargin )
```

This procedure uses the solution U for interpolating the virtual distorted area. The values of the solution are rescaled from 0 to 1. It forces to 1 the values on the boundary marked with *BdFlags* and to 0 the others boundary. This function can be used for performing a harmonic interpolation.

The arguments are:
BdFlags: The flags of the virtually moved boundary.
U: The solution.
Mesh: The Mesh data structure.
varargin: The options described above (If we use a sparse matrix this procedure become very slow).

Harmonic Interpolation

```
function Mesh = eval_Displacement_Harmonic_Generic( Mesh , ...
                VM_BdFlag , VM_BdVal , BdVal , ...
                F_HANDLE , GD_HANDLE , varargin )
```

Perform a harmonic interpolation by defining and computing a new FEM problem. This function can be also used for the partial harmonic procedure.

This function assign an user defined Dirichlet boundary condition (VM_BdVal) to the virtually moved boundary (marked with VM_BdFlag) and another Dirichlet boundary conditions (BdVal) to the others boundary. It interpolate only the positives values of u.

The arguments are:
Mesh: The Mesh data structure.
VM_BdFlag: The flags of the virtually moved boundary.
VM_BdVal: The Dirichlet boundary conditions of the virtually moved boundary.
BdVal: The Dirichlet boundary conditions of the fixed boundary.
F_HANDLE: Right hand side source term (use: @f_LShap)
GD_HANDLE: Dirichlet boundary function (use: @g_D_LShapGeneric)
varargin: The options described above.

Boundary Layers

```
function Mesh = eval_Displacement_BDLayers(Mesh,BdFlags,L,varargin)
```

Perform the boundary layers interpolation described in the chapter 4.2.

The arguments are:
Mesh: The Mesh data structure.
BdFlags: The flags of the virtually moved boundary.
L: The numbers of layers involved in the interpolation.
varargin: The options described above.

Exponential around a circle

```
function Mesh = eval_Displacement_Exp(Mesh,R1,C1,R2,varargin)
```

Execute the exponential interpolation described in the chapter 4.5. The procedure set to 1 the values that lie on the circle with radius R1 and center C1 and rescale the others values with the procedure described in 4.5. The arguments are:
Mesh: The Mesh data structure.
R1: The radius of the inner circle.
C1: The center of the inner circle.
R2: Trhe radius of the outer circle.
varargin: The options described above.

Linear around a circle

```
function Mesh = eval_Displacement_Linear_Generic( Mesh , R1 , C1 , R2 ,
```

Perform the conic interpolation described in the chapter 4.5. The values on the R1 are set to 1 and the others values are rescaled with the algorithm described in the chapter 4.5.

The arguments are:
Mesh: The Mesh data structure.
R1: The radius of the inner circle.
C1: The center of the inner circle.
R2: The radius of the outer circle.
C2: The center of the outer circle.
varargin: The options described above.

Exponential around a generic shape

```
function Mesh = eval_Displacement_Exp_Generic( Mesh , BdFlags , RI , RL
```

Perform the exponential interpolation procedure described in the chapter 4.4.

The arguments are:
Mesh: The Mesh data structure.
BdFlags: The flags of the virtually moved object.
RI: An user defined parameter.
RLim: The size of the shell.
varargin: The options decribed above.

Linear around a generic shape

```
function Mesh = eval_Displacement_Linear_Generic_Shape( Mesh , BdFlags , RL
```

Perform the exponential interpolation procedure described in the chapter 4.4.

The arguments are:
Mesh: The Mesh data structure.
BdFlags: The flags of the virtually moved object.
RI: An user defined parameter.
RLim: The size of the shell.
varargin: The options described above.

Coarsed displacement interpolation

```
function Mesh = virtual_Translation_MultiLevel(S,U,BdFlags,n,nref,MeshStart
```

Perform the coarsed displacement interpolation described in the chapter 4.6. Given the initial Mesh data structure (MeshStart) it refine the Mesh for n steps.
Displace the nodes along the vector S by rescaling the values in U.
Continue the refinement process up to nref.

The arguments are:
S: The direction of the displacement. The magnitude of S is used as a maximal translation.
U: The interpolation rule, for example a solution.
BdFlags: The flags of the virtually moved object.
n: The number of refinement after which the displacement process is performed. *nref*: The maximal number of refinement.
MeshStart: The original mesh data structure.
Mesh: The mesh data structure.
options: The option described above. In the others functions are stored in the 'varargin' argument.
varargin: Should store the handler to the functions for projecting the new edges in MeshStart

to the boundary:

```
Mesh = virtual_Translation_MultiLevel(S,[],-2,3,...
    NREFS,MeshStart,Mesh,'sc', ...
    DHANDLE,{Center  R},-1 , DHANDLE,{Center2  R2},-2, ...
    DHANDLE,{Center  R},-1 , DHANDLE,{Center2+S  R2},-2) ;
```

The first set of functions is used before the displacement and the second set after.
Each function is described by three arguments: DHANDLE,Center R,-1
DHADLE are the handler to the functions, the list ... are the arguments and the last number is the flag of the boundary.

A.3. Others functions

```
function  New_Mesh = refine_REG_SB(Old_Mesh,varargin)
```

It performs one regular red refinement step on the MESH structure. During red refinement the signed distance functions DHANDLE (function handle/inline object) are used to project the new vertices on the boundary edges onto the boundary of the domain marked with the BDFLAG. This function accept any arbitrary number of DHANDLE,DPARAMS,BDFLAG triples.

Example:

```
Mesh = refine_REG(Mesh,@dist_circ,{[0 0] 1},-1);
Mesh = refine_REG(Mesh, ...
       @dist_circ,{[0 0] 1},-1,...
       @dist_circ,{[0 1] 1},-2);
```

```
function U = Test_Solution( Mesh )
```

Compute the analytical solution of the problem described in the chapter 5.2.

A.4. Start a test

The followings routines run a single test on a configuration:

```
main_LFE_DQUAD.m
main_LFE_D_CIRCLES.m
main_LM.m
main_LFE_NSq.m
```

For changing the number of refinements and the intepolation procdedure see the first lines of the code.

For starting a full test we must use the test functions:
These functions write all possible charts and information in a directory specified with the variable 'dir_name' (as a default log directory we use 'esp'). If you want to organize the test in a subdirectory structure comment out the line '`dir_name = 'esp' ;`' and the results will be written in a subdirectory structure where the main directory is the 'testname' and the subdirectory is the name of the interpolation (for the maximal number of ref. and the test procedure see the firsts lines of the code).

The test function are:
`Test_VirtualWork_2.m`: Start the comparison with the analitical solution.
`Test_VirtualWork.m`: Start the test with the translated inner circle.
`Test_VirtualWorkLM.m`: Start the a test with the LM structure.
`Test_VirtualWorkLM_APosteriori.m`: Start the a test with the LM structure by performing an adaptive refinement.
`Test_VirtualWork_Duo.m`: Perform the test on the structure with two inner circles.
`Test_VirtualWork_SQ.m`: Test with an inner off-centres square.
`Test_VirtualWork_SQ_APosteriori.m`: Test with an inner off-centred square and a posteriori refinement.
`Test_VirtualWork_NSq_APosteriori.m`: Perform the with two nested squares and the BC described the new problem sheet.

Bibliography

[Cou83] J. L. Coulomb. A methodology for the detewination of global electromechanical quantities froy a finite element analysis and its application to the evaluation of magnetic forces, torques and stiffness. *IEEE Transactions on magnetics*, Mag. 19(6):2514–2519, 1983.

[Hen04] Francois Henrotte. The eggshell approach for the computation of electromagnetic forces in 2d and 3d. *The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, 23(4):996–1005, 2004.