Title: Numerical solution of PDEs in periodical domain

Author: Jordi Barceló Mercader

Advisor: Sonia Fernández Méndez and David Codony Gisbert

Department: Departament d'Enginyeria Civil i Ambiental (751)

Academic year: 2017-2018
Numerical solution of PDEs in periodical domain

Jordi Barceló Mercader

Supervised by Sonia Fernández Méndez and David Codony Gisbert
January, 2018
Abstract

We present in this work two schemes of approximation for numerical solutions of PDEs. The first one is the maximum entropy method (max-ent) and the second one is the b-spline method. These methods let us impose a special kind of boundary conditions: periodic boundary conditions for unbounded domains. Some experiments need a large domain (or unbounded domain), however, this domain is divided into some periodic cells. We develop a technique that let us simulate in the whole domain only doing a simulation in one cell. We apply this method for the resolution of second and fourth order problems (with periodic boundary conditions) like: Laplace, Kirchhoff plate and flexoelectricity.

Keywords

Max-ent, b-splines, periodicity, Laplace, Kirchhoff plate, flexoelectricity
1. Introduction

Solving numerically a partial differential equation usually involves using the finite element method (FEM) and some boundary conditions such as Neumann’s conditions or Dirichlet’s conditions. Nevertheless, with such a method some difficulties may arise (as it will be shown later). If we need to solve a fourth order PDE we will need to impose two conditions on the same boundary, for example, the value of the function and the value of the normal gradient of the function. Then, we should use a mixed formulation that despite having some problems with stability, can be solved. If we look for a 2D solution that is periodic in one direction, we would impose a special boundary condition, that means that the value at 2 boundaries is the same, like a cylinder. Furthermore, we seek a solution of class $C^\infty$, so we cannot impose the value; but we need to impose continuity of all the derivatives too. Such a problem is impossible to avoid using FEM.

Instead of using FEM, we are going to work with 2 different methods without elements. The first one is the maximum entropy method, which was developed in [1] and the second one uses b-splines as shape functions and was developed in [2]. Max-ent is a method that uses global shape functions, the functions are defined in the entire domain, it does not use elements, but it needs cells for integration. B-spline method is quite different because the shape functions are defined by a knot vector (as it will be explained in section 2.3). By implementing this method we also lose the bijection between shape functions and points or particles. When we use FEM, we know that the weight of each shape function within an element is equal to a value of one node, with b-splines this condition is no longer true, so we need to change our way of thinking.

The goal of this thesis is to solve the PDE associated with the flexoelectricity problem with the periodic boundary conditions explained before. Flexoelectricity is a coupling between strain gradient and electric potential, rather than strain and polarization as is piezoelectricity. The kind of materials that are used for flexoelectricity have some periodic cells, like a thin film, so if we can implement a good method for imposing the boundary conditions, we would avoid simulating the entire domain (we will only need to simulate in one cell). Apart from solving the flexoelectricity equations, the periodic boundary conditions can be useful when using metamaterials because of their symmetries.

Before solving the final problem, we need to solve some easier problems to test the code. Firstly, we solve the Laplace equation with periodic boundary conditions, however, this is an equation of second order only. To extend out testing to some more complicated cases, we continue with the Kirchhoff plate equation that it is a fourth order equation and, finally, we tackle the flexoelectricity problem.
2. Laplace equation

The Laplace equation is a problem about diffusivity (as the heat equation), our domain is unbounded and periodic in $x$ direction which means that we are looking for a solution that is periodic on $x$. The strong form of the problem is:

$$
\begin{cases}
-\Delta u(x) = f & x \in \mathbb{R}/[0,1)\times[0,1] \\
u(x) = g & x \in \mathbb{R}/[0,1)\times\{0\} \\
u(x) = h & x \in \mathbb{R}/[0,1)\times\{1\}
\end{cases}
$$

As we will use functions $g$ and $h$ that are from class $C^\infty$, we want a solution in the same class over the whole domain: this means that the solution must be infinitely differentiable at $x = 0$. We can see an example of the domain in figure 1. Now we need to get the weak form of the problem. The derivation of the weak form which we present below is in Appendix A.

Find $u$ such that $u(x) = g$ if $x \in \mathbb{R}/[0,1)\times\{0\}$ and $u(x) = h$ if $x \in \mathbb{R}/[0,1)\times\{1\}$ and satisfies:

$$
\int_{\Omega} \nabla v \cdot \nabla u \, d\Omega = \int_{\Omega} vf \, d\Omega
$$

for all $v$ such that $v = 0$ on the Dirichlet Boundary.

![Figure 1: Domain simulated (left), same colour means same particle in our domain and an example of a shape function extracted from [1]](image)

2.1 Maximum entropy approximation

As previously emphasized, instead of using a partition of the domain in elements, we are going to use shape functions that are defined in a huge part of the domain (check [1] for using local and global shape functions), the discretization of the solution is:

$$
u^h(x) = \sum_{a=1}^N p_a(x) u_a$$
Numerical solution of PDEs in periodical domain

where $N$ is the number of nodes in which we want to know the value of the solution. We can use the nodes of a typical square mesh, or we can use any unstructured nodes that we want. Nonetheless, we require 2 conditions for the shape functions:

$$\sum_{a=1}^{N} p_a(x)x_a = x \quad \sum_{a=1}^{N} p_a(x) = 1$$

These conditions guarantee that linear functions are exactly reproduced. Those shape functions can be thought of as distribution functions. They are actually negative exponentials located on the nodes (figure 1), and can be computed solving a non-linear system, we are not going to go into the detail about this method but they can be traced in [1].

Since we do not have a mesh, we need to create some cells to do the integration, because we cannot integrate over the whole domain. After the discretization and the change of the integral by a sum in integration points, we get the following equation:

$$\sum_{\text{cells}} \sum_{i \in \text{IP}} \left( \sum_{a=1}^{N} (w_i \nabla p_j(x) \cdot \nabla p_a(x) u_a) = \sum_{\text{cells}} \sum_{i \in \text{IP}} p_j(x) f w_i \right)$$  \hspace{1cm} (1)

As we can see, first we sum over all the cells and then we sum over all the integration points inside the cell. Using the previous equation, we can get a system of equation without any boundary conditions applied. However, we actually want to apply the periodicity condition.

2.2 Implementation of periodic boundary condition

The key point of the implementation is the definition of phantom nodes and real nodes. Real nodes are those nodes that are in the region we want to integrate, in our case it is the square $[0,1) \times [0,1]$. Phantom nodes are those which are outside that region, for example the node $(0.5,0.5)$ would be a real node and the node $(1.5,0.5)$ would be a phantom node. Each phantom node is related to a real node, the one which is congruent with. The two previous nodes of the example are related. Each real node is related to with an infinite number of phantom nodes. However, we do not need many phantom nodes but only those needed for the shape function. In our case, we simulate only nodes between $-1$ and $2$ for $x$ coordinate as we can see in figure 1.

That definition is taken into account when we assemble the matrix of the system (1), instead of adding some value to a phantom node, we will add it to the real node associated with that phantom node. Doing that, we can force the periodic behaviour of the solution (we can think of it as rolling a cylinder). We get a system of equations of the form $Ku = f$, and we will reduce it to enforce Dirichlet boundary conditions. Then we apply all the boundary conditions so we can solve the system using a solver and get the result.

To test the code we are going to use as a Dirichlet boundary condition: $u(x,y) = y \sin(2\pi x)$. In figure 2, we can see the solution we obtain with a degree of freedom equal to 88. We can observe that the solution is actually a periodic solution, and we get optimal convergence (slope equals to 2). One of the problems of max-ent is that we cannot easily get shape functions for a higher order (as explained in [1]). We should look for another suitable method.
2.3 B-Spline approximation

B-spline approximation is another approximation that can be easily computed using a recursive formula, but, first, we need to define the knot vectors. A knot vector is an increasing and finite list of numbers \( \{ u_0, u_1, ..., u_m \} \), for example \( \{ 0, 1, 2, 3, 4, 5 \} \). These numbers are called knots. The b-splines are piece-wise polynomials and they are defined using a knot vector as:

\[
N_{i,0}(u) = \begin{cases} 
1 & \text{if } u_i \leq u < u_{i+1} \\
0 & \text{otherwise}
\end{cases}
\]

\[
N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)
\]

The first sub index of the b-spline function is the number of the shape function and the second one is the degree of the b-spline. When we talk about the degree of the b-spline, it means that the b-splines can reproduce polynomials of the same degree. One remark from the previous expression is that we can get the value of the b-splines at a point by only doing linear combinations of b-splines with less degree. This is useful because, instead of solving a non-linear system as in max-ent, all we have to do is some basic operations. Another important thing is that a b-spline of degree p has continuity up to the p-1 derivative. Such continuity can be reduced if we repeat a knot in the knot vector. We reduce the continuity in a point as many times as we repeat that knot. We can see an example of the b-splines of degree 2 of the knot vector \( \{ 0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5 \} \) in figure 3. For more details about the properties of b-splines check [2].

We need 2D shape functions, however the previous ones were 1D shape functions. Our shape functions are created doing the tensor product of 1D shape functions. As we want to impose Dirichlet boundary conditions in the y direction we use a knot vector of the form \( \{ 0, 0, 0, u_1, u_2, ..., 1, 1, 1 \} \), with as many 0 and 1 as the degree we want to obtain plus 1. Doing that we get shape functions that are only continuous.
Numerical solution of PDEs in periodical domain

Figure 3: Example of b-splines of degree 2 with the knot vector: \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}, image extracted from [2](left) and an example of b-splines of degree 2 used in the periodical axe, the big lines represents the real shape functions and the finest ones represent some phantom shape functions, and shape functions with the same colours represent the same shape function.

on the boundary, so we can apply the boundary conditions in a strong way. We can use others knot vectors but then, we should apply some techniques as Nietzche’s method or Lagrange multiplier to apply the boundary conditions. As we want periodicity in x direction, we need a knot vector where all the shape functions are equal, therefore we use a knot vector with some phantom knots (knots outside the domain) at the beginning and at the end. We can see an example of the b-splines of degree 2 of the knot vector \{-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4\} in figure 4. This is an example of a knot vector we should use in x direction.

2.4 Imposing periodic boundary condition

Using the previous shape functions we can get a discretization of the problem, but we need to apply the periodicity of the solution in x direction. To enforce periodicity we are going to relate shape functions as we have done before with max-ent, but, in this case, we are not going to relate particles. In figure 3 we can see some shape functions and we can relate the biggest ones with the finest ones because each colour means different shape functions. During the assembly of the system matrix, instead of adding some values to shape functions that it is fine, we add the contribution to the shape function that is bigger. If we consider that these shape functions are defined in a circular way, it is obvious that some shape functions (same colour) will overlap. After applying the periodicity condition of the solution, we can reduce the system to impose the Dirichlet boundary condition and solve the system as before.

To test the code we use the same boundary conditions as before, the convergence plot of the L2 error for degree of b-splines equal to 1,2,3,4 can be seen in figure 4. We get optimal convergence, slope \(p + 1\). After solving the problem with max-ent and b-splines, we compare them. In figure 4 there is an additional plot comparing b-splines and max-ent using the same degree of approximation. Max-ent has smaller error than b-splines for the same degree of freedom, but as we can easily implement the b-splines for a higher order, we can actually get less error than max-ent. Since b-splines satisfies all the conditions we need and the way of computing the value of the shape functions is less problematic, we will use b-splines to solve the fourth order equation.
3. Kirchhoff plate

In the next step, we use b-spline to find the solution of a fourth order problem. We will solve a Kirchhoff plate [3], for which the strong form of the problem is: find \( w \) such that \( w \) satisfies:

\[
\begin{aligned}
\nabla \cdot \nabla \cdot m &= f \quad \text{on } \Omega \\
w &= g \quad \text{on } \Gamma_D \\
n^+ \cdot \nabla w &= \theta_n \quad \text{on } \Gamma_D
\end{aligned}
\]  

(2)

where:

\[ m_{ij} = D (\nu \delta_{ij} w_{kk} + (1 - \nu) w_{ij}) \]

We assume that our domain is the same as in section 2, and the load is:

\[ f = \frac{25}{2} \sin(\pi(x + 0.25)) \cos(\pi(x + 0.25)) \sin(\pi(y + 0.5)) \]

This problem has an analytical solution that is:

\[ w(x, y) = \frac{1}{4\pi^4 D} \sin(2\pi(x + 0.25)) \sin(\pi(y + 0.5)) \]

Where \( D = (Ez^3)/(12(1 - \nu^2)) \) is the isotropic bending stiffness [4]. \( z = 0.01 \) is the thickness of the plate, \( E = 100 \) is the young modulus and \( \nu = 0.3 \) is the Poisson’s ratio. The boundary conditions have been calculated from the analytical solution at each point of the boundary. Now we need to derive the weak form of the problem (2), see appendix A. The weak form is: find \( w \) such that \( w = g \) and \( n^+ \cdot \nabla w = \theta_n \) on \( \Gamma_D \) and satisfies:
Numerical solution of PDEs in periodical domain

\[ \int_{\Omega} \nabla \nabla v : m \, d\Omega = \int_{\Omega} vf \, d\Omega \]

for all \( v \) such that \( v = 0 \) and \( n^+ \cdot \nabla v = 0 \) on \( \Gamma_D \). We slightly change the way of imposing the Dirichlet boundary conditions: we impose them using a least square method on the boundary, instead of fixing the value of the shape functions at the knots.

For the test we are going to compare the numerical solution with the analytical one using the L2 norm to compute the L2 error. We can see in figure 5 the numerical solution that we get and the convergence rate of the L2 error. We obtain optimal convergence rate for degree 3 and 4.

Figure 5: Numerical solution of the Kirchhoff plate, it simulates \( w(x, y) = \frac{1}{4\pi D} \sin(2\pi(x+0.25)) \sin(\pi(y+0.5)) \) (left) and the convergence plot of L2 error for b-splines of degree 3 and 4 (right).

4. Flexoelectricity

Flexoelectricity is a coupling between strain gradient and electric field, instead of strain and electric field as is piezoelectricity. The main advantage of flexoelectricity is that we can generate it in all materials, whereas piezoelectricity can only be generated in piezoelectric materials. The way flexoelectricity works is straightforward: suppose that we have a square cell, when we apply a strain to a material we change the form of the square but if the strain is uniform we will not be able to create a dipole as it happens in the first picture of the figure 6. However, if we apply a non-uniform strain, like a bending, the positive and negative charges will separate and we will create a dipole (as we can see in the second picture in figure 6).

In our consideration of the full model, we solve flexoelectricity in a piezoelectric material, so then, we have both contributions. We use the plain stress simplification to convert the 3D problem into a 2D problem, and we use the value for the operators of [5][6].

The first thing we need is the bulk enthalpy density [5][6]:

\[ H_B = \frac{1}{2} \varepsilon_{ijkl} \varepsilon_{ij} \varepsilon_{kl} + \frac{1}{2} h_{ijklm} \varepsilon_{ij,k} \varepsilon_{lm,n} - \frac{1}{2} K_{lm} E_l E_m - e_{ij} E_l \varepsilon_{ij} - \mu_{ijk} E_l \varepsilon_{ij,k} \]

where:
The first term of the enthalpy corresponds to the elasticity model and depends on the elasticity tensor $C$, the second term corresponds to the strain-gradient elasticity model and depends on the strain-gradient elasticity tensor $h$, the third one corresponds to the linear electric model, which depends on the dielectricity tensor $K$. The fourth one corresponds to the linear piezoelectric coupling that couples strain tensor with electric field using the piezoelectricity tensor $e$ and the last term is the flexoelectricity one that couples the strain-gradient tensor with the electric field using the flexoelectricity tensor $\mu$.

Knowing that, we can formulate the strong form of the flexoelectricity problem as: find $u$ such that $u$ satisfies:

\[
\begin{align*}
\nabla \cdot (\hat{\sigma} - \nabla \cdot \tilde{\sigma}) + b &= 0 \quad \text{on } \Omega \\
\nabla \cdot \hat{D} - q &= 0 \quad \text{on } \Omega \\
\n\nabla u \cdot n^+ &= g \quad \text{on } \Gamma_D \\
\phi &= h \quad \text{on } \Gamma_D
\end{align*}
\]

where:

\[
\hat{\sigma}_{ij} = \frac{\partial H_B}{\partial \varepsilon_{ij}} = C_{ijkl} \varepsilon_{kl} - e_{ij} E_l
\]

\[
\tilde{\sigma}_{ijk} = \frac{\partial H_B}{\partial \varepsilon_{ij,k}} = h_{ijklmn} \varepsilon_{lm,n} - \mu_{ijk} E_l
\]

\[
\hat{D}_l = -\frac{\partial H_B}{\partial E_l} = K_{lm} E_m + e_{ij} \varepsilon_{ij} + \mu_{ijk} \varepsilon_{ij,k}
\]

From equation (3) we can get the weak form of the problem (see Appendix A). The weak form is: find $u$ and $\phi$ such that satisfy the Dirichlet boundary conditions and:
Numerical solution of PDEs in periodical domain

\[ \int_{\Omega} \nabla w \cdot \hat{D} \, d\Omega + \int_{\Omega} w q \, d\Omega = 0 \]
\[ \int_{\Omega} \nabla \nabla v \cdot \tilde{\sigma} \, d\Omega + \int_{\Omega} \nabla v : \hat{\sigma} \, d\Omega - \int_{\Omega} \nu \cdot b \, d\Omega = 0 \]

for all \( v \) and \( w \) such that \( v = 0 \) on \( \Gamma_D \), \( \nabla v \cdot n^+ = 0 \) on \( \Gamma_D \) and \( \phi = 0 \) on \( \Gamma_D \).

4.1 Numerical results

Now we are going to compare the numerical and analytical solutions. The analytical one we seek is:

\[ u_x = 0.2 \sin(2\pi x) \]
\[ u_y = 0.2 \sin(\pi y/2) \]
\[ \phi = 0.2 \sin(2\pi x) \sin(\pi y/2) \]

we are going to use the Dirichlet boundary conditions equal to the analytical solution on the boundary. Doing that and solving the system as before, we get the solution we can see in figure 7. We can also observe that the convergence is optimal: we conclude that the code works well and it is able to reproduce any problem.

The main codes of the three problems: Laplace, Kirchhoff plate and flexoelectricity are posted in the appendix B together with the codes that compute the matrix of the system.

5. Conclusions

In this work we worked successfully with 2 different approximations of the solution, max-ent and b-spline. Although both methods solve the Laplace equation without any problems, we have decided the b-spline is more suitable for solving PDE of fourth order. The reasons already mentioned before are: b-splines are computed recursively from b-spline of smaller degree whereas max-ent shape functions are computed solving a non-linear system; b-spline can be easily implemented for higher order and max-ent has some problems with the implementation for higher order. The last reason is not trivial and has high significance: despite the fact that we have used \( C^1 \) continuity, we might need more continuity and with b-splines there is no need to change the method to calculate the shape functions.

We have imposed periodicity of the solution in a way that it does not depend on the place where we want the periodicity, in other words, no matter where we cut the domain, we are going to get the same solution with a translation on space. The method works perfectly with the problems we have tested: Laplace, Kirchhoff plate and flexoelectricity.

Finally, we have solved the flexoelectricity problem and we have got optimal convergence. Notice that flexoelectricity is a very important phenomenon because it can be reproduced in all kinds of materials.
Figure 7: From left to right and from up to down, numerical $u_x$ computed, numerical $u_y$ computed, numerical $\phi$ computed and convergence plot of L2 error of flexoelectricity.

Although flexoelectricity is an important property, it had not been investigated until recently (10-15 years ago) so it is a topic worth further research.

There are some limitations in this thesis that could be minimised. Firstly, we have imposed only periodicity in one direction because we wanted to create a thin film, but it could be used for simulating a big plane with multiple equal cells (we would apply periodicity in both directions). The method can also be used for simulating a big lattice in 3D space. In this thesis we have not worked with the 3D case because of its computational cost. The second thing we could improve is about the boundary conditions: here we have applied Dirichlet Boundary conditions, but we could use Neumann conditions that are more suitable when imposing stresses instead of strains.

The last improvement has to do with the geometry of the domain. In our case, we have used a square domain but we could use any periodic shape. We could integrate an interface that does not coincide with the boundary of the domain, but we would need to apply some techniques as Nitsche’s method and an interface defined by a level set functions.
6. Bibliography

References


A. Derivation of weak form

A.1 Laplace problem

The strong form of the problem is: find $u$ such that $u$ satisfies:

$$
\begin{align*}
-\Delta u(x) &= f & x &\in \mathbb{R}/[0,1)\times[0,1] \\
u(x) &= g & x &\in \mathbb{R}/[0,1)\times\{0\} \\
u(x) &= h & x &\in \mathbb{R}/[0,1)\times\{1\}
\end{align*}
$$

Firstly, we multiply the equation by a test function $v$ and we integrate over the whole domain:

$$
\int_\Omega v(x)(-\Delta u(x)) \, d\Omega = \int_\Omega v(x)f \, d\Omega
$$

Now we apply integration by parts:

$$
\int_\Omega \nabla v(x) \cdot \nabla u(x) \, d\Omega - \int_{\partial\Omega} v(x)\nabla u(x) \cdot n^+ \, dl = \int_\Omega v(x)f \, d\Omega
$$

Since the test function $v$ banishes on the Dirichlet boundary, we get the final result: find $u$ such that $u(x) = g$ if $x \in \mathbb{R}/[0,1)\times\{0\}$ and $u(x) = h$ if $x \in \mathbb{R}/[0,1)\times\{1\}$ and $u$ satisfies:

$$
\int_\Omega \nabla v \cdot \nabla u \, d\Omega = \int_\Omega vf \, d\Omega
$$

for all $v$ such that $v = 0$ on the Dirichlet Boundary.

A.2 Kirchhoff plate

The strong form of the problem is: find $w$ such that $w$ satisfies:

$$
\begin{align*}
\nabla \cdot \nabla \cdot m &= f & \text{on } \Omega \\
w &= g & \text{on } \Gamma_D \\
^+ \cdot \nabla w &= \theta_n & \text{on } \Gamma_D
\end{align*}
$$

Firstly, we multiply the equation by a test function $v$ and we integrate over the whole domain:

$$
\int_\Omega v \nabla \cdot \nabla \cdot m \, d\Omega = \int_\Omega vf \, d\Omega
$$

Now we apply integration by parts:

$$
-\int_\Omega \nabla v \cdot (\nabla \cdot m) \, d\Omega + \int_{\partial\Omega} vn^+ \cdot \nabla \cdot m \, dl = \int_\Omega vf \, d\Omega
$$

Since $v$ banishes on the boundary, we can erase the second term and we can apply integration by parts in the first term again:
Numerical solution of PDEs in periodical domain

\[
\int_{\Omega} \nabla \nabla v : m \, d\Omega + \int_{\partial \Omega} \nabla v \cdot m \cdot n^+ \, dl = \int_{\Omega} vf \, d\Omega
\]

Since \( \nabla v = 0 \) we can erase the second term and finally we get the weak form of the problem: find \( w \) such that \( w = g \) and \( n^+ \cdot \nabla w = \theta_n \) on \( \Gamma_D \) and \( w \) satisfies:

\[
\int_{\Omega} \nabla \nabla v : m \, d\Omega = \int_{\Omega} vf \, d\Omega
\]

for all \( v \) such that \( v = 0 \) and \( n^+ \cdot \nabla v = 0 \) on \( \Gamma_D \).

A.3 Flexoelectricity problem

The strong form of the problem is: find \( u \) and \( \phi \) such that both satisfy:

\[
\begin{cases}
\nabla \cdot (\hat{\sigma} - \nabla \cdot \tilde{\sigma}) + b = 0 & \text{on } \Omega \\
\nabla \cdot \hat{D} - q = 0 & \text{on } \Omega \\
u = f & \text{on } \Gamma_D \\
\nabla u \cdot n^+ = g & \text{on } \Gamma_D \\
\phi = h & \text{on } \Gamma_D
\end{cases}
\]

Firstly, we multiply the second equation by a test function \( w \) and we integrate over the whole domain:

\[
\int_{\Omega} w \cdot (\nabla \cdot \hat{D}) \, d\Omega - \int_{\Omega} wq \, d\Omega = 0
\]

We apply integration by parts:

\[
-\int_{\Omega} \nabla w \cdot \hat{D} \, d\Omega + \int_{\partial \Omega} w \cdot \hat{D} \cdot n^+ \, dl - \int_{\Omega} wq \, d\Omega = 0
\]

Since \( w \) banishes on the boundary:

\[
\int_{\Omega} \nabla w \cdot \hat{D} \, d\Omega + \int_{\Omega} wq \, d\Omega = 0
\]

Now we multiply the first equation by a test function \( v \) and we integrate over the whole domain:

\[
\int_{\Omega} v \cdot \nabla \cdot (\hat{\sigma} - \nabla \cdot \tilde{\sigma}) \, d\Omega + \int_{\Omega} v \cdot b \, d\Omega = 0
\]

We apply integration by parts:

\[
-\int_{\Omega} \nabla v \cdot \hat{\sigma} \, d\Omega + \int_{\Omega} \nabla v \cdot (\nabla \cdot \tilde{\sigma}) \, d\Omega + \int_{\partial \Omega} v \cdot (\hat{\sigma} - \nabla \cdot \tilde{\sigma}) \cdot n^+ \, dl + \int_{\Omega} v \cdot b \, d\Omega = 0
\]

Since \( v = 0 \) on the boundary, we can erase the third term and we can apply integration by parts in the second term:
\[-\int_{\Omega} \nabla v : \hat{\sigma} \, d\Omega - \int_{\Omega} \nabla \nabla v : \tilde{\sigma} \, d\Omega + \int_{\partial\Omega} \nabla v : \tilde{\sigma} \cdot n^+ \, dl + \int_{\Omega} v \cdot b \, d\Omega = 0\]

Since \(\nabla v = 0\) on the boundary:

\[\int_{\Omega} \nabla \nabla v : \tilde{\sigma} \, d\Omega + \int_{\Omega} \nabla v : \hat{\sigma} \, d\Omega - \int_{\Omega} v \cdot b \, d\Omega = 0\]

So, the weak form is: find \(u\) and \(\phi\) such that satisfy the Dirichlet boundary conditions and:

\[\int_{\Omega} \nabla w \cdot \tilde{D} \, d\Omega + \int_{\Omega} w q \, d\Omega = 0\]

\[\int_{\Omega} \nabla \nabla v : \tilde{\sigma} \, d\Omega + \int_{\Omega} \nabla v : \hat{\sigma} \, d\Omega - \int_{\Omega} v \cdot b \, d\Omega = 0\]

for all \(v\) and \(w\) such that \(v = 0\) on \(\Gamma_D\), \(\nabla v \cdot n^+ = 0\) on \(\Gamma_D\) and \(\phi = 0\) on \(\Gamma_D\).
Numerical solution of PDEs in periodical domain

B. Codes in matlab

Main code and computation of matrices for Laplace equation with max-ent:

1. \texttt{addpath(path,'functions');}
2. \texttt{clear all}
3. \texttt{L = [3 1];}
4. \texttt{degree=2; \%Degree of integration quadrature}
5. \texttt{n=5; \%Number of integration cells,}
6. \texttt{Nx = 3*n+1;}
7. \texttt{Ny = n+1;}
8. \texttt{centre = [0.5*L(1); 0.5*L(2)];}
9. \texttt{%creation of the particles}
10. \texttt{[x_nodes hx hy] = NonUniformGrid2D(Nx, Ny, L(1), L(2), centre);}
11. \texttt{nPts = size(x_nodes,1);}
12. \texttt{[phan, real] = phantomnodes(x_nodes); \%Creation of real and phantom nodes}
13. \texttt{%The vector phan relate real nodes with phantom nodes}
14. \texttt{h = max(hx, hy);}
15. \texttt{% Definition of locality parameters}
16. \texttt{gamma = 3;}
17. \texttt{beta = gamma/h/h;}
18. \texttt{beta_n = ones(1, nPts)*beta;}
19. \texttt{sourceFunction=@(x,y) 4*pi*pi*sin(2*pi*x)*y;}
20. \texttt{bound=@(x,y) y*sin(2*pi*x);}
21. \texttt{solution=@(X) X(2)*sin(2*pi*X(1));}
22. \texttt{% Sample points (where the shape functions are computed)}
23. \texttt{Div=2*n;}
24. \texttt{[IP, IW] = createIntegrationPoints(Div, degree);}
25. \texttt{IP = IP + [ones(size(IP, 1), 1) zeros(size(IP, 1), 1)];}
26. \texttt{%Define boundary nodes and boundary values}
27. \texttt{ids = (1:nPts)'};
28. \texttt{boundary = []';}
29. \texttt{for i=1:nPts}
30. \texttt{eps=1e-8;}
31. \texttt{x=x_nodes(i,1);}
32. \texttt{y=x_nodes(i,2);}
33. \texttt{if (\((\text{abs}(y)<\text{eps}) \&\&(\text{abs}(y-1)<\text{eps})\)) \&\&((x+eps > 1) \&\&(x+eps < 2))}
34. \texttt{boundary = [boundary; i];}
35. \texttt{end}
36. \texttt{end}
notbb = setdiff(real, boundary);

bvalue = [];

for i=1:size(boundary)
    bvalue = [bvalue; bound(x_nodes(boundary(i),1), x_nodes(boundary(i),2))];
end

% Options for the computation of the shape functions
dim = 2;
options.dim = 2; % spatial dimension
options.verb = 0; % information and plots: 0=OFF, 1=ON
options.TolNR = 1.e-8; % Newton-Raphson Tolerance
options.Tol0 = 1.e-6; % Target Zero Tolerance
options.beta_n = beta_n; % locality parameters
range_n = zeros(nPts,1);
for i=1:nPts
    range_n(i) = max(2*h, sqrt(-log(options.Tol0) ./ beta_n(i)));
end
options.range_n = range_n; % range definition

% Shape functions ans its gradients are computed
[p_s, dp_s, s_nears] = lme_wrapper(x_nodes, IP, options);

% Computing the matrices of the system
[K, f] = computeSystemLaplacePhantom(IP, IW, p_s, dp_s, s_nears, sourceFunction, x_nodes, phan);

% Imposing Dirichlet Boundary conditions
realbb = encuentra(real, boundary);
realnotbb = encuentra(real, notbb);
sol = K(realnotbb, realnotbb) \ (f(realnotbb) - K(realnotbb, realbb) * bvalue);
complete = zeros(size(real, 1), 1);
complete(realbb) = bvalue;
complete(realnotbb) = sol;
total = complete(encuentra(real, phan));
error = L2error(IW, IP, solution, p_s, total, s_nears); % Calculate the L2 error

% Plot of the solution
tri = delaunay(x_nodes(:,1), x_nodes(:,2)); % x, y, z column vectors
trisurf(tri, x_nodes(:,1), x_nodes(:,2), total, 'Edgecolor', 'none', 'Facecolor', 'interp');

function [K, f] = computeSystemLaplacePhantom(IP, IW, ps, dp_s, s_nears, sourceFunction, x_nodes, phan)
Numerical solution of PDEs in periodical domain

```matlab
nOfNodes = size(x_nodes,1);
K=spalloc(nOfNodes,nOfNodes,nOfNodes*5); %global matrix initialization
f=zeros(nOfNodes,1);

%Loop in elements
for i=1:size(IP,1)
    GaussPoint=IP(i,:);
    GaussWeight=IW(i);
    N=ps{i}';
    Gk = dp{s{i}}';
    Te=s_nears{i}; %nodes in the element
    K(phan(Te),phan(Te))=K(phan(Te),phan(Te))+Gk'*Gk*GaussWeight; %
    %assembly using the vector phantom
    f(phan(Te)) = f(phan(Te)) + sourceFunction(GaussPoint(1),GaussPoint(2))*N'*GaussWeight;
end

Main code and computation of matrices for Laplace equation with b-splines:

close all
clear;

degree=3; %Degree of integration quadrature
n=43; %Number of cells!!
Nx = n+1;
Ny = n+1;

[phan, real]=phantommod(n, degree);

% Definition of locality parameters
sourceFunction=@(x,y) 4*pi*pi*sin(2*pi*x)*y;
bound=@(x,y) y*sin(2*pi*x);
solution=@(X) X(2)*sin(2*pi*X(1));

%Define boundary functions
bb=[1:n (n*(n+degree-1)+1):(n*(n+degree))]';
bv=boundaryvalue(n, degree, bound);
notbb=setdiff(1:real,bb)';
[IP, IW]=gausspoints2D(degree);

%Compute system
[K, f]=computesystemlaplaceperiodic(IP, IW, sourceFunction, n, phan, degree);
sol=K(notbb, notbb)
```

complete=zeros(real,1);
complete(bb)=bv;
complete(notbb)=sol;
total=complete(phan(:));

[x_nodes solut]=realsolution(degree,n,total);

tri = delaunay(x_nodes(:,1),x_nodes(:,2)); %x,y,z column vectors
trisurf(tri,x_nodes(:,1),x_nodes(:,2),solut,'Edgecolor','none','Facecolor','interp');

%error=L2error(IP,IW,solution,n,total,degree)
%error=L2error(IW,IP,solution,p_s,total,s_nears)

function [K, f]=computesystemlaplaceperiodic(IP,IW,sourceFunction,n,phan,p)

IW=IW/4/n/n;
IP=IP/n;
l=size(IP,1);
nOfNodes=max(phan);

%Creation of knot vectors
knotx=-p:n+p;
knoty=knotx;
for i=1:p
    knoty(i)=0;
    knoty(n+p+1+i)=n;
end
knotx=knotx/n;
knoty=knoty/n;
knot={[knotx,knoty]};

%Definition of matrices
K=spalloc(nOfNodes,nOfNodes,nOfNodes*10); %global matrix initialization
f=zeros(nOfNodes,1);

%Loop in cells
for i=1:n
    for j=1:n
        IPoints=IP+[(i-1)*ones(l,1)/n zeros(l,1)]+[zeros(l,1) (j-1)*ones(l,1)/n];
        for k=1:size(IP,1)
            GaussPoint=IPoints(k,:);
            GaussWeight=IW(k);
            N=BSplines_RecursiveBasisFunctions(GaussPoint,p,[p+i,p+j],

19
Numerical solution of PDEs in periodical domain

\[ G_k = \text{BSplines RecursiveBasisFunctions}(\text{GaussPoint}, p, [p+i, p+j], \text{knot}, 1) ; \]
\[ \text{Te=cerca}([i+p, j+p], p, n) ; \% \text{nodes in the element} \]
\[ K(\text{phan}(\text{Te}), \text{phan}(\text{Te})) = K(\text{phan}(\text{Te}), \text{phan}(\text{Te})) + G_k G_k' \text{GaussWeight} ; \]
\[ f(\text{phan}(\text{Te})) = f(\text{phan}(\text{Te})) + \text{sourceFunction(}\text{GaussPoint}(1), \text{GaussPoint}(2)) \text{N*GaussWeight} ; \]

Main code and computation of matrices for Kirchhoff plate with b-splines:

```matlab
close all
clear;
global D nu

z=0.01;
E=100;
nu=0.3;
D=(E*z^3)/(12*(1-nu^2));
degree=3; \% Degree of integration quadrature
n=21; \% Number of cells!!
Nx = n+1;
Ny = n+1;
\% Creation of phantom vector
[phan, real]=phantonmod(n, degree);

\% Definition of locality parameters
sourceFunction=@(x,y) 25/2*cos(\pi*(x+0.25))*sin(\pi*(x+0.25))*sin(\pi*(y+0.5));
bound=@(x,y) 1/4/\pi/\pi/\pi/\pi/D*sin(2*\pi*(x+0.25))*sin(\pi*(y+0.5));
boundary=@(x,y) 1/4/\pi/\pi/\pi/\pi/D*sin(2*\pi*(x+0.25))*cos(\pi*(y+0.5));
solution=@(X) 1/4/\pi/\pi/\pi/\pi/D*sin(2*\pi*(X(1)+0.25))*sin(\pi*(X(2)+0.5));

\% Define boundary functions and the value of the boundaries (2 values)
bb=[1:2*n (n*(n+degree-2)+1):(n*(n+degree))] ;
bv=boundaryvalue2(n, degree, bound, boundary);
notbb=setdiff(1:real, bb) ;
[IP, IW]=gausspoints2D(degree);
```
%Compute system
[K, f] = computesystemplate(IP, IW, sourceFunction, n, phan, degree, D, nu);
%Reduce the system and solve the system
sol = K(notbb, notbb) \ (f(notbb) - K(notbb, bb)*bv);

complete = zeros(real, 1);
complete(bb) = bv;
complete(notbb) = sol;
total = complete(phan(:));

%Compute solution on the knots
[x_nodes solut] = realsolution(degree, n, total);
true = [];
for i = 1: size(x_nodes, 1)
    true = [true; solution(x_nodes(i, :))];
end

%Plot the solution
tri = delaunay(x_nodes(:, 1), x_nodes(:, 2)); % x, y, z column vectors
trisurf(tri, x_nodes(:, 1), x_nodes(:, 2), solut, 'Edgecolor', 'none', 'Facecolor', 'interp');

%Compute the L2 error
error = L2error(IP, IW, solution, n, total, degree);

function [K, f] = computesystemplate(IP, IW, sourceFunction, n, phan, p, D, nu)
IW = IW/4/n/n;
IP = IP/n;
l = size(IP, 1);
nOfNodes = max(phan);

%Creation of knot vectors
knotx = p:n+p;
knoty = knotx;
for i = 1:p
    knoty(i) = 0;
knoty(n+p+1+i) = n;
end
knotx = knotx/n;
knoty = knoty/n;
knot = {knotx, knoty};

%Definition of matrices
K = spalloc(nOfNodes, nOfNodes, nOfNodes*10); % Global matrix initialization
f = zeros(nOfNodes, 1);

%Loop in cells
for i = 1:n
    for j = 1:n
Numerical solution of PDEs in periodical domain

IPoints = IP + [(i - 1) * ones(l, 1) / n, zeros(l, 1)] + [zeros(l, 1), (j - 1) * ones(l, 1) / n];

for k = 1:size(IP, 1)
    GaussPoint = IPoints(k, :);
    GaussWeight = IW(k);
    N = BSplines_RecursiveBasisFunctions(GaussPoint, p, [p+i, p+j], knot, 0);
    Gk = BSplines_RecursiveBasisFunctions(GaussPoint, p, [p+i, p+j], knot, 1);'
    Hk = BSplines_RecursiveBasisFunctions(GaussPoint, p, [p+i, p+j], knot, 2);'
    Te = cerca([i+p, j+p], p, n); %nodes in the element
    elemental = zeros(size(Te, 1), size(Te, 1)); %Elemental matrix
    for Tt = 1:size(Te, 1)
        for Tr = 1:size(Te, 1)
            mat1 = [Hk(Tt, 1) Hk(Tt, 3); Hk(Tt, 3) Hk(Tt, 2)];
            mat2 = D*[nu*Hk(Tr, 2) + Hk(Tr, 1) (1-nu)*Hk(Tr, 3); (1-nu)*Hk(Tr, 3) nu*Hk(Tr, 1) + Hk(Tr, 2)];
            elemental(Tt, Tr) = sum(sum(mat1 * mat2));
        end
    end
    K(phan(Te), phan(Te)) = K(phan(Te), phan(Te)) + elemental * GaussWeight; %assembly
    f(phan(Te)) = f(phan(Te)) + sourceFunction(GaussPoint(1), GaussPoint(2)) * N * GaussWeight;
end
end
end

Main code and computation of matrices for flexoelectricity equation with b-splines:

close all
clear;
addpath('tprod');
degree = 3; %Degree of integration quadrature
n = 5; %Number de cells !!
Nx = n + 1;
Ny = n + 1;
[phan2, real] = phantonmod(n, degree);
maxi = size(phan2, 1);
phan = [phan2; phan2 + real; phan2 + 2 * real];
% Definition of locality parameters
x = sym('x'); y = sym('y');
u1ana = 0.2 * sin(2 * pi * x);
u1anay = diff(u1ana, y);
u2ana = 0.2 * sin(pi * y/2);
u2anay = diff(u2ana, y);
phiana = 0.2 * sin(2 * pi * x) * sin(pi * y/2);
phianay = diff(phiana, y);
bana = giveb(u1ana, u2ana, phiana); % Compute functions b and q given a solution
qana = giveq(u1ana, u2ana, phiana);
u1 = @(X,Y) subs(u1ana, [x,y], [X,Y]);
u1y = @(X,Y) subs(u1anay, [x,y], [X,Y]);
u2 = @(X,Y) subs(u2ana, [x,y], [X,Y]);
u2y = @(X,Y) subs(u2anay, [x,y], [X,Y]);
phi = @(X,Y) subs(phiana, [x,y], [X,Y]);
phiy = @(X,Y) subs(phianay, [x,y], [X,Y]);
b = @(X,Y) subs(bana, [x,y], [X,Y]);
q = @(X,Y) subs(qana, [x,y], [X,Y]);

% Define boundary functions
bb0 = [1:n (n*(n+degree-1)+1):(n*(n+degree))]';
bb1 = [1:2*n (n*(n+degree-2)+1):(n*(n+degree))]';
bb = [bb1; bb1+real; bb0+2*real];
bv1 = boundaryvalue2(n, degree, u1, u1y);
bv2 = boundaryvalue2(n, degree, u2, u2y);
bv3 = boundaryvalue(n, degree, phi, phiy);
bv = [bv1 bv2 bv3];
notbb = setdiff(1:3*real, bb)';
[IP, IW] = gausspoints2D(degree);

% Compute system
[K, f] = computesystemflexo(IP, IW, n, phan, degree, b, q, maxi);
% Reduce and solve the system
sol = K(notbb, notbb) \ (f(notbb) - K(notbb, bb)*bv);
complete = zeros(3*real, 1);
complete(notbb) = bv;
complete(notbb) = sol;
total = complete(phan(:));

% Plot solution (one coordinate)
[x_nodes solut] = realsolution(degree, n, total(1:maxi));
true = [];
Numerical solution of PDEs in periodical domain

for \( i = 1: \text{size}(x\_nodes, 1) \)
\[
\text{true} = [\text{true}; \text{solution}(x\_nodes(i, :))];
\]
end

\( \text{tri} = \text{delaunay}(x\_nodes(:, 1), x\_nodes(:, 2)); \) \%x, y, z column vectors
\( \text{trisurf} (\text{tri}, x\_nodes(:, 1), x\_nodes(:, 2), \text{solut}, '\text{Edgecolor}', '\text{none}', '\text{Facecolor}', '\text{interp}'); \)
\%Compute the error
\( \text{error} = \text{L2error}(\text{IP}, \text{IW}, \text{solution}, n, \text{total}, \text{degree}) \)

function \([K, f] = \text{computesystemflexo}(\text{IP}, \text{IW}, n, \text{phan}, p, b, q, \text{maxi})\)
\( \text{IW} = \text{IW}/4/n/n; \)
\( \text{IP} = \text{IP}/n; \)
\( l = \text{size}(\text{IP}, 1); \)
\( \text{nOfNodes} = \max(\text{phan}); \)
\( \text{op} = \text{operators}; \)

\%Creation of knot vectors
\( \text{knotx} = -p:n+p; \)
\( \text{knoty} = \text{knotx}; \)
for \( i = 1:p \)
\( \text{knoty}(i) = 0; \)
\( \text{knoty}(n+p+1+i) = n; \)
end
\( \text{knotx} = \text{knotx}/n; \)
\( \text{knoty} = \text{knoty}/n; \)
\( \text{knot} = \{\text{knotx}, \text{knoty}\}; \)

\%Definition of matrices
\( \text{K} = \text{spalloc}(\text{nOfNodes}, \text{nOfNodes}, \text{nOfNodes} \times 10); \) \%global matrix initialization
\( \text{f} = \text{zeros}(\text{nOfNodes}, 1); \)

\%Loop in cells
for \( i = 1:n \)
for \( j = 1:n \)
\( \text{IP} = \text{IP} + [(i-1) \star \text{ones}(1, 1) / n \text{zeros}(1, 1)] + [\text{zeros}(1, 1) \text{zeros}(j-1) \star \text{ones}(1, 1) / n]; \)
for \( k = 1: \text{size}(\text{IP}, 1) \)
\( \text{GaussPoint} = \text{IP}(k, :); \)
\( \text{GaussWeight} = \text{IW}(k); \)
\( \text{N} = \text{BSplines}_{\text{RecursiveBasisFunctions}}(\text{GaussPoint}, p, [p+i, p+j], \text{knot}, 0); \)
\( \text{Gk} = \text{BSplines}_{\text{RecursiveBasisFunctions}}(\text{GaussPoint}, p, [p+i, p+j], \text{knot}, 1); \)
\( \text{Hk} = \text{BSplines}_{\text{RecursiveBasisFunctions}}(\text{GaussPoint}, p, [p+i, p+j], \text{knot}, 2); \)
\( \text{Te} = \text{cerca}([i+p, j+p], p, n); \) \%nodes in the element
for Tt=1:size(Te,1)
    for Tr=1:size(Te,1)

%Definition of the 3 coordinates of each shape
%function
pointstest=[Te(Tt) Te(Tt)+maxi Te(Tt)+2*maxi];
pointsreal=[Te(Tr) Te(Tr)+maxi Te(Tr)+2*maxi];

%Functions of test
eps1t=[Gk(Tt,1) 1/2*Gk(Tt,2);1/2*Gk(Tt,2) 0];
gradeps1t(:,:,1)=[Hk(Tt,1) 1/2*Hk(Tt,3);1/2*Hk(Tt,3) 0];
gradeps1t(:,:,2)=[Hk(Tt,3) 1/2*Hk(Tt,2);1/2*Hk(Tt,2) 0];
eps2t=[0 1/2*Gk(Tt,1);1/2*Gk(Tt,1) Gk(Tt,2)];
gradeps2t(:,:,1)=[0 1/2*Hk(Tt,1);1/2*Hk(Tt,1) Hk(Tt,3)];
gradeps2t(:,:,2)=[0 1/2*Hk(Tt,3);1/2*Hk(Tt,3) Hk(Tt,2)];
Et=-[Gk(Tt,1);Gk(Tt,2)];

%Functions of real
eps1r=[Gk(Tr,1) 1/2*Gk(Tr,2);1/2*Gk(Tr,2) 0];
gradeps1r(:,:,1)=[Hk(Tr,1) 1/2*Hk(Tr,3);1/2*Hk(Tr,3) 0];
gradeps1r(:,:,2)=[Hk(Tr,3) 1/2*Hk(Tr,2);1/2*Hk(Tr,2) 0];
eps2r=[0 1/2*Gk(Tr,1);1/2*Gk(Tr,1) Gk(Tr,2)];
gradeps2r(:,:,1)=[0 1/2*Hk(Tr,1);1/2*Hk(Tr,1) Hk(Tr,3)];
gradeps2r(:,:,2)=[0 1/2*Hk(Tr,3);1/2*Hk(Tr,3) Hk(Tr,2)];
Er=-[Gk(Tr,1);Gk(Tr,2)];

%Elemental matrix (each gauss point)
elemental(1,1)=tprod(gradeps1t,[-1 1 2 3],tprod(op.h,[1 3 1 -2 -3],tprod(gradeps1r,[-1 1 2 3]),[-1 -2 -3]));
elemental(1,2)=tprod(gradeps1t,[-1 1 -2 3],tprod(op.h,[1 2 3 -1 2 -3],tprod(gradeps2r,[-1 1 -2 3]),[-1 -2 -3]));
elemental(1,3)=tprod(gradeps1t,[-1 1 -2 3],tprod(op.m,[1 2 3 -1 2 -3],Er,[-1]),[-1 -2 -3]));
elemental(2,1)=tprod(gradeps2t,[-1 1 -2 3],tprod(op.h,[1 2 3 -1 2 -3],tprod(gradeps1r,[-1 1 -2 3]),[-1 -2 -3]));
elemental(2,2)=tprod(gradeps2t,[-1 1 -2 3],tprod(op.h,[1 2 3 -1 2 -3],tprod(gradeps1r,[-1 1 -2 3]),[-1 -2 -3]));
elemental(2,3)=tprod(gradeps2t,[-1 1 -2 3],tprod(op.h,[1 2 3 -1 2 -3],tprod(gradeps1r,[-1 1 -2 3]),[-1 -2 -3]));
elemental(2,2)=tprod(gradeps2t,[-1 -2 -3],tprod(op.h,[1 2 3 -1 -2 -3],gradeps2r,[-1 -2 -3]),[-1 -2 -3])+tprod(eps2t,[-1 -2],tprod(op.C,[1 2 -1 -2],
eps2r,[-1 -2]));

elemental(2,3)=tprod(gradeps2t,[-1 -2 -3],tprod(op.
mu,[-1 1 2 3],Er,[-1]),[-1 -2 -3])-tprod(eps2t,[-1 -2],tprod(op.e,[-1 1 2],Er,[-1]),[-1 -2]);

elemental(2,3)=tprod(gradeps2t,[-1 -2 -3],tprod(op.
mu,[-1 1 2 3],Er,[-1]),[-1 -2 -3])-tprod(eps2t,[-1 -2],tprod(op.e,[-1 1 2],Er,[-1]),[-1 -2]);

elemental(2,3)=tprod(gradeps2t,[-1 -2 -3],tprod(op.
mu,[-1 1 2 3],Er,[-1]),[-1 -2 -3])-tprod(eps2t,[-1 -2],tprod(op.e,[-1 1 2],Er,[-1]),[-1 -2]);

elemental(3,1)=tprod(Et,[-1],tprod(op.e,[1 -1 -2],
eps1r,[-1 -2]),[-1])+tprod(Et,[-1],tprod(op.mu,[1 -1 -2 -3],gradeeps1r,[-1 -2 -3]),[-1]);

elemental(3,2)=tprod(Et,[-1],tprod(op.e,[1 -1 -2],
eps2r,[-1 -2]),[-1])+tprod(Et,[-1],tprod(op.mu,[1 -1 -2 -3],gradeeps2r,[-1 -2 -3]),[-1]);

elemental(3,3)=tprod(Et,[-1],tprod(op.K,[1 -1 ],Er ,[-1 ]),[-1]);

end

f(phan(pointstest)) = f(phan(pointstest)) + felem*GaussWeight;

end

end

end