UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Title of the master thesis

Subtitle (optional)

Dissertation presented by
Firstname LASTNAME , Firstname LASTNAME

for obtaining the Master's degree in
Speciality

Supervisor(s)
Firstname LASTNAME, Firstname LASTNAME

Reader(s)
Firstname LASTNAME, Firstname LASTNAME , Firstname LASTNAME

Academic year 20..-20..

Chapter 1

Theory

This is the intro.

1.1 Presentation of the problem

In this section, we will present the problem we will solve. Let us consider a domain $\Omega \in \mathbb{R}^2$ with its boundary $\Gamma = \partial\Omega$. Then, we want to find a function $u : \Omega \to \mathbb{R}$ that satisfies :

$$\nabla^2 u = f \qquad \text{on } \Omega \tag{1.1}$$

$$u = u_0 \qquad \text{on } \Gamma \tag{1.2}$$

Where $f : \Omega \to \mathbb{R}$ and $u_0 : \Gamma \to \mathbb{R}$ are two given functions. We also assume that f and u_0 satisfy the standard regularity assumptions. Let us denote $H^1(\Omega)$, the Sobolev space containing functions whose partial derivatives up to order 1 are in $L^2(\Omega)$. Let us also define $H_0^1(\Omega)$, a subspace of $H^1(\Omega)$ where the functions are equal to zero on Γ.

$$H_0^1(\Omega) = \{v \in H^1(\Omega), v|_\Gamma = 0\}$$

Let us then multiply equation 1.1 by any function v in $H_0^1(\Omega)$ and integrate it over the domain. And since the laplacian is equivalent to the divergence of the gradient, we have that :

$$\int_\Omega \nabla \cdot (\nabla u) v \, dxdy = \int_\Omega f v \, dxdy \tag{1.3}$$

Any function u that is a solution of 1.1 is obviously also a solution of 1.3 for every function $v \in H_0^1(\Omega)$. Let us now recall that for any function u and v regular enough :

$$\nabla \cdot (v \nabla u) = \nabla u \cdot \nabla v + v \nabla \cdot (\nabla u)$$

Using this relation on equation 1.3, we get :

$$\int_\Omega \nabla u \cdot \nabla v \, dxdy = -\int_\Omega \nabla \cdot (v \nabla u) dxdy - \int_\Omega f v \, dxdy \tag{1.4}$$

Finally, using the divergence theorem, we have :

$$\int_\Omega \nabla u \cdot \nabla v \, dxdy = -\int_\Gamma (v \nabla u) \cdot \mathbf{n} \, ds - \int_\Omega f v \, dxdy \tag{1.5}$$

Since $v \in H_0^1(\Omega)$, it is equal to zero on the boundary and therefore :

$$\int_\Omega \nabla u \cdot \nabla v \, dxdy = -\int_\Omega fv \, dxdy \tag{1.6}$$

We can now state the weak formulation of problem 1.1. Let us assume, without loss of generality that $u_0 = 0$. Then, we want to find a function $u \in H_0^1(\Omega)$ that satisfies :

$$\int_\Omega \nabla u \cdot \nabla v \, dxdy = -\int_\Omega fv \, dxdy \qquad \forall v \in H_0^1(\Omega) \tag{1.7}$$

1.2 Spectral element method on non conforming meshes

Equation 1.7 is the problem we will be attempting to solve. However, the equation must still hold for every function $v \in H_0^1(\Omega)$. This section presents the method used for the discretization of the problem, namely the spectral element method on a non conforming mesh.

The first part of the task is the discretization of the domain Ω. As explained in the introduction, we will use non conforming meshes. So we will present such meshes, present their particularities and explain how we will handle them.

The second part of the task is the discretization of equation 1.7. The method here presented is the spectral element method, which consists of using high degree piece-wise polynomials as basis functions. We will first show how to obtain the linear system of equation ($Au = b$) that arises with this method. We will afterwards present the one dimensional basis functions used and the location of the nodes on the 1D reference element. We will then explain how to construct the global basis functions in the presence of hanging nodes. We will finally show how to compute the right hand side of the linear system and how to perform an efficient matrix-vector product without having to explicitly construct the matrix A.

1.2.1 Discretization of the domain

Let us first tackle the discretization of the domain. In classical AMR, the mesh depends on the problem to solve and the refinement is performed to obtain the fact that the error committed on each quadrant is roughly the same (ref here). But that is not the focus of the work done here. Instead, let us assume that we already have a mesh G consisting of unstructured quadrilaterals (denoted Ω_e) and that might be non conforming, i.e. where we have the presence of hanging nodes. As always, we have :

$$G = \bigcup_e \Omega_e$$
$$\Omega_i \cap \Omega_j = \emptyset \qquad \text{if } i \neq j$$

We can define a hanging node as being a node that is not shared between all its neighboring quadrants. We can note that hanging nodes can only be found on edges. Figure 1.1 shows an example of a mesh containing hanging nodes. We can see that the vertices in red are hanging nodes since they are vertices of the smaller quadrants neighboring them but not of the bigger quadrants neighboring them. Those nodes will require a special treatment in the spectral element method.

We have however one restriction to make on the mesh. We will allow two neighboring quadrants to have only one level of difference in the refinement. This means that the number of neighbors of a quadrant through a given edge is maximum two. We call such meshes 1-irregular. Figure 1.2 shows an example of an illegal non conforming mesh. We can see that on that mesh

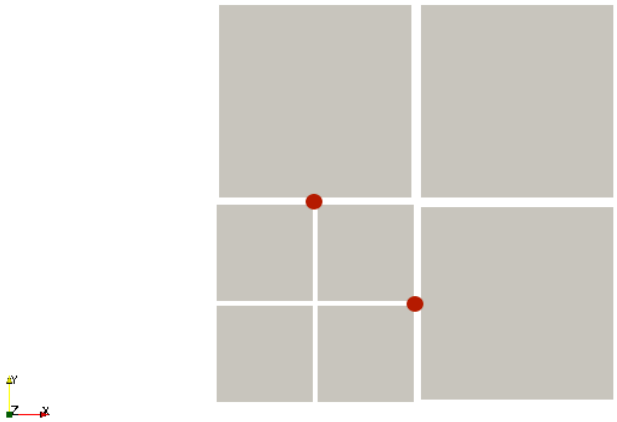


Figure 1.1: Example of a legal non conforming mesh (it is 1-irregular). We can see that we have two hanging nodes in red since they are not vertices of the bigger quadrants neighboring them.

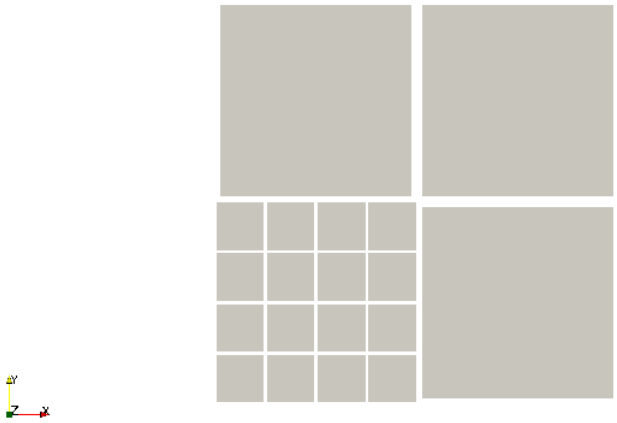


Figure 1.2: Example of an illegal non conforming mesh (it is 2-irregular). We can see that the top left quadrant has four neighbors through its south edge while we imposed a maximum of two.

the top left quadrant has four neighbors through its south edge. Since the maximum difference between the level of refinement between two neighboring quadrants is two, such a mesh is called 2-irregular. Handling all possibilities of hanging faces for a general k-irregular mesh (with $k > 1$) is impossible and that is why we (and the vast majority of others) consider as illegal meshes such as the one presented in figure 1.2.

For the rest of this chapter, it is important to note that, in a 1-irregular mesh, a hanging edge (an edge containing hanging nodes) is always half of a non hanging edge for its neighboring quadrant.

Once we have our mesh, it is clear that we can compute the integral of any scalar function g as the sum of integrals on the quadrants. Formally, we have :

$$\int_\Omega g \, dxdy = \sum_e \int_{\Omega_e} g \, dxdy \tag{1.8}$$

All the integrals we compute will be performed quadrant by quadrant and then summed.

1.2.2 Discretization of equation 1.7

Since equation 1.7 must hold for every function $v \in H_0^1(\Omega)$, it is not very practical. We will therefore restrict ourselves to $V^h \subset H_0^1(\Omega)$, where V^h has a finite dimension. Let us denote the functions that form the basis of V^h by $\phi_1, ..., \phi_N$ (therefore, the space V^h is of dimension N). We will define explicitly those basis functions later.

Using Galerkin approximation, we will also restrict our search of the solution to V^h. Let us denote this solution u^h. Therefore, we have that :

$$u^h = \sum_{j=1}^{N} u_j \phi_j$$

Asking equation 1.7 to be satisfied for every function in V^h is equivalent to ask it to be satisfied for every function of its basis. As a results, the discretized version of equation 1.7 is given by :

$$\sum_{j=1}^{N} \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, dxdy \, u_j = -\int_\Omega f\phi_i \, dxdy \qquad \text{for } i = 1, ..., N \tag{1.9}$$

Solving this to get the nodal values u_j is actually solving a linear system of equations $\mathbf{Au} = \mathbf{b}$ where we define :

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, dxdy \tag{1.10}$$

$$b_i = -\int_\Omega f\phi_i \, dxdy \tag{1.11}$$

Those integral will be performed using equation 1.8.

1.2.3 Basis functions on the reference element and Gauss-Lobatto-Legendre nodes

As shows equations 1.10 and 1.11, we need to compute integrals to solve the problem. Here, we will perform the integration using Gauss-Lobatto-Legendre (GLL) quadrature. The particularity of 1D GLL nodes is that they include the end points of the interval on which we compute the integral. For example, on the one dimensional reference element, it means that the GLL nodes

Number of nodes $p+1$	Points ξ_i	Weights w_i
2	± 1	1
3	± 1	$\frac{1}{3}$
	0	$\frac{4}{3}$
4	± 1	$\frac{1}{6}$
	$\pm\sqrt{\frac{1}{5}}$	$\frac{5}{6}$
5	± 1	$\frac{1}{10}$
	$\pm\sqrt{\frac{3}{7}}$	$\frac{49}{90}$
	0	$\frac{32}{45}$

Table 1.1: Values of the GLL nodes and the associated weights on the reference one dimensional element for the first few degrees.

include $\xi = -1$ and $\xi = 1$. This particularity enables us to use the GLL nodes both as quadrature points and as global nodes where we compute our numerical solution. We will see later that this allows us to compute the matrix-vector product very efficiently.

For $p+1$ GLL nodes, we can integrate exactly polynomials of degree at most $2p-1$. It is less than for the Gauss-Legendre quadrature (which achieve to integrate exactly polynomials up to order $2p+1$). The way to compute the GLL nodes and their weights is for example detailled (ref here). Table 1.1 shows the values for different degrees. We can see that the nodes are not uniformly distributed but they tend to cluster near the end points $\xi = -1$ and $\xi = 1$.

Let us denote the $p+1$ GLL nodes on the reference 1D element by ξ_0, ξ_1, ..., ξ_p. Since the GLL nodes contain the end points of the interval, we can also use them as global nodes. We will use Lagrangian polynomials for the interpolation. Let us denote them $l_0(\xi)$, $l_1(\xi)$, ... , $l_p(\xi)$. Let us recall that :

$$l_i(\xi) = \prod_{j=0, j\neq i}^{p} \frac{\xi - \xi_j}{\xi_i - \xi_j}$$

For the following parts of this chapter, it is very important to note that :

$$l_i(\xi_j) = \delta_{ij}$$

Where δ_{ij} is to be understood as the Kronecker delta. This fact will allow us later to compute the matrix-vector product in a very efficient way.

Let us also use this opportunity to define the derivation matrix H as :

$$H_{ij} = l_i'(\xi_j) \tag{1.12}$$

Let us now define the basis functions on the 2D reference element. In 2D, we will use $(p+1)^2$ nodes, tensor product of the 1D GLL nodes. If we denote the 1D GLL nodes by ξ_i^1, the 2D nodes are thus located at :

$$(\xi_i, \eta_j) = (\xi_i^1, \xi_j^1) \qquad \text{for } i, j = 0, 1, ..., p$$

On the reference quadrant $\xi, \eta \in [-1; 1]$, we have in the same way that the basis function associated with node I located at (ξ_i, η_j) is given the tensor product of the 1D basis functions :

$$\phi_I(\xi, \eta) = l_i(\xi) l_j(\eta)$$

Thus, if we want to obtain the value of a field u in the reference quadrant, whose value at node I is given by u_{ij}, we interpolate as :

$$u(\xi,\eta) = \sum_{i=0}^{p}\sum_{j=0}^{p} u_{ij} l_i(\xi) l_j(\eta)$$

1.2.4 Global basis function

Let us first mention the distinction between global and local nodes. Each quadrant has $(p+1)^2$ local GLL nodes but that does not mean that all of them are global nodes, since some might be hanging. One solution consists to treat hanging nodes as global nodes and then add equations in the linear system $Au = b$ to enforce the continuity of the numerical solution. Another solution is to immediately enforce the continuity of the numerical solution by having continuous global basis function. This is what is done here. We will consider N global nodes (that are the GLL nodes for at least one quadrant) and give them a global unique index $I = 1, ..., N$.

montrer mesh ici et aussi montrer une shape function pour p=2

We want our global basis function ϕ_I to be a piece-wise polynomial of order p and to have a value of exactly 1 at global node I and to be equal to 0 on any other global node.

In order to define such a function, we first need a mapping that goes from the 2D reference element to the actual quadrant e, i.e. $x^e(\xi,\eta)$ and $y^e(\xi,\eta)$ that for quadrant e give the actual coordinates of any point in the reference element. Let us also define the inverse mappings $\xi^e(x,y)$ and $\eta^e(x,y)$.

The last thing we need is is a global to local operator that gives for local node (ξ_i, η_j) on quadrant e the weight of global node I. Let us denote this operator $R^e_{ij}(I)$. In a conforming mesh, since each local node correspond to a global node, $R^e_{ij}(I)$ can only be equal to 0 or to 1. In a non conforming mesh, however, the value of $R^e_{ij}(I)$ must be computed to ensure the continuity of the global basis function. For example, if we have hanging nodes on the east edge of quadrant e (where $i = p$), then $R^e_{pj}(I)$ is given by :

$$R^e_{pj}(I) = l_k\left(\frac{\xi_j}{2} \pm \frac{1}{2}\right) \qquad \text{if } I \text{ is the } k\text{-th global node on the non hanging east edge}$$

More explanation about how to compute the operator $R^e_{ij}(I)$ can be found ref here. The plus or minus sign in the above formula comes from the fact that a hanging edge can be the first or the second part of the larger non hanging edge. This is why we accept only 1-irregular meshes : because we only have to handle two cases when we compute $R^e_{ij}(I)$. If we accepted k-irregular meshes with $k > 1$, we would have too many different cases to look at.

With this operator and our mapping, it is then possible to compute the value of the basis function associated with node I inside any element e. It is given by :

$$\phi_I(x,y) = \sum_{i=0}^{p}\sum_{j=0}^{p} R^e_{ij}(I)\; l_i(\xi^e(x,y))\; l_j(\eta^e(x,y)) \qquad \text{if } (x,y) \in \Omega_e \tag{1.13}$$

Figure 1.3 shows a global basis function on the mesh presented in figure 1.1 for $p = 1$. We can see that it is not entirely trivial since the global node associated with the basis function has a influence in quadrants where we have hanging nodes. We can also see that it is indeed a continuous function that is a piece-wise bilinear polynomial.

Let us also note that there is no global basis function associated with a hanging node.

In practice, we do not build the operator $R^e_{ij}(I)$ explicitly since only nodes located on edges can be hanging.

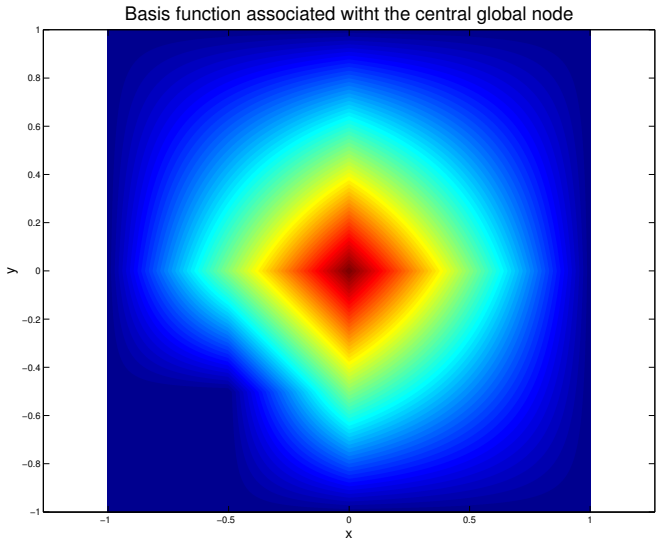


Figure 1.3: Global basis function associated with the global GLL node located in the center of the mesh presented in figure 1.1 for $p = 1$.

1.2.5 Computing the right-hand side

Now that we have defined our global basis functions, it is possible to perform the integration needed to compute b_j (equation 1.11). As explained before, we will perform the integration quadrant by quadrant. Let us denote :

$$b_I^e = -\int_{\Omega_e} f\phi_I \, dxdy$$

We obviously have that $b_I = \sum_e b_I^e$. Let us also define the reference element associated with quadrant e by $\hat{\Omega}_e$ and the jacobian of the mapping as :

$$J^e(\xi,\eta) = \begin{pmatrix} \frac{\partial x^e}{\partial \xi} & \frac{\partial x^e}{\partial \eta} \\ \frac{\partial y^e}{\partial \xi} & \frac{\partial y^e}{\partial \eta} \end{pmatrix}$$

Then the integration on quadrant e becomes :

$$b_I^e = -\int_{\hat{\Omega}_e} f(x^e(\xi,\eta), y^e(\xi,\eta))\phi_I(x^e(\xi,\eta), y^e(\xi,\eta))|J^e(\xi,\eta)| \, d\xi d\eta$$

Using equation 1.13, we get :

$$b_I^e = -\int_{\hat{\Omega}_e} f(x^e(\xi,\eta), y^e(\xi,\eta)) \left(\sum_{i=0}^{p}\sum_{j=0}^{p} R_{ij}^e(I) l_i(\xi) l_j(\eta) \right) |J^e(\xi,\eta)| \, d\xi d\eta$$

This is where the fact that our global nodes are also GLL nodes becomes important. To alleviate a little the formulas, let us define two new notations : $J^e(\xi_m,\eta_n) = J_{mn}^e$ and $f(x^e(\xi_m,\eta_n), y^e(\xi_m,\eta_n)) = f_{mn}^e$. Using the GLL quadrature rule, we have :

$$b_I^e = -\sum_{m=0}^{p}\sum_{n=0}^{p} w_m w_n f_{mn}^e \left(\sum_{i=0}^{p}\sum_{j=0}^{p} R_{ij}^e(I) l_i(\xi_m) l_j(\eta_n) \right) |J_{mn}^e|$$

Since $l_i(\xi_j) = \delta_{ij}$, this becomes :

$$b_I^e = -\sum_{m=0}^{p}\sum_{n=0}^{p} w_m w_n f_{mn}^e R_{mn}^e(I)|J_{mn}^e|$$

This expression looks more complicated than it actually is. Indeed, the operator $R_{mn}^e(I)$ is often equal to zero. It is non zero only if in quadrant e, the local node (m,n) is the global node I (in which case it is equal to 1) or if the local node (m,n) is located on a hanging edge where the global node I has an influence. Therefore, in a quadrant where we do not have hanging nodes, and if the local node (m,n) has I as global number, we have :

$$b_I^e = -w_m w_n f_{mn}^e |J_{mn}^e|$$

1.2.6 Computing a matrix-vector product

As said in the introduction, we will use an iterative method to solve the system presented by equations 1.10 and 1.11, namely the preconditioned conjugate gradients. As a result, we do not need to build the matrix A explicitly but rather we need to be able to perform a matrix-vector product Au where u is a vector containing the values of our numerical solution at the global nodes, i.e. u_I is the value of the solution at global node I. By linearity of the integral and the derivative, and the definition of u^h, we have :

$$\begin{aligned}(Au)_I &= \sum_{J=1}^{N}\int_\Omega \nabla\phi_I \cdot \nabla\phi_J \, dxdy \, u_J \\ &= \int_\Omega \nabla\phi_I \cdot \nabla\left(\sum_{J=1}^{N}\phi_J u_J\right) dxdy \\ &= \int_\Omega \nabla\phi_I \cdot \nabla u^h \, dxdy\end{aligned}$$

Just as in the previous subsection, we will compute this integral quadrant by quadrant. The first thing we want to do is to compute the value of u^h at the local GLL nodes on quadrant e. Let us define, u_{mn}^e as this local value :

$$u_{mn}^e = \sum_{I=1}^{N} R_{mn}^e(I) u_I$$

Let us then define u^e as the restriction of u^h on quadrant e. Since we know that, on quadrant e, u^h must be a polynomial of degree p going through the $(p+1)^2$ GLL local nodes whose values are given by u_{mn}^e for $m,n = 0,...,p$, and since we know that such a polynomial is unique, we have :

$$u^e(x,y) = \sum_{m=0}^{p}\sum_{n=0}^{p} u_{mn}^e l_m(\xi^e(x,y)) l_n(\eta^e(x,y)) \qquad \text{if } (x,y) \in \Omega_e$$

Using our restriction, the linearity of the different operators and the equation 1.13, we have :

$$\begin{aligned}(Au)_I^e &= \int_{\Omega_e} \nabla\phi_I \cdot \nabla u^h \, dxdy \\ &= \int_{\Omega_e} \nabla\phi_I \cdot \nabla u^e \, dxdy \\ &= \sum_{m=0}^{p}\sum_{n=0}^{p} R_{mn}^e(I) \int_{\Omega_e} \nabla\left(l_m(\xi^e(x,y)) l_n(\eta^e(x,y)\right) \cdot \nabla u^e \, dxdy \qquad (1.14)\end{aligned}$$

Let us now focus on the last part of equation 1.14. Since we want to integrate on the reference element, we first have to take care of the derivatives. To simplify the notations, let us define : $L_{mn}(\xi,\eta) = l_m(\xi)l_n(\eta)$. Then, using the chain rule, we have :

$$\begin{aligned}\int_{\Omega_e} \nabla\left(l_m(\xi^e(x,y))l_n(\eta^e(x,y)\right)\cdot\nabla u^e\,dxdy = \int_{\hat{\Omega}_e} &\left(\frac{\partial L_{mn}}{\partial\xi}\frac{\partial\xi}{\partial x}+\frac{\partial L_{mn}}{\partial\eta}\frac{\partial\eta}{\partial x}\right)\left(\frac{\partial u^e}{\partial\xi}\frac{\partial\xi}{\partial x}+\frac{u^e}{\partial\eta}\frac{\partial\eta}{\partial x}\right)|J^e| \\ &+\left(\frac{\partial L_{mn}}{\partial\xi}\frac{\partial\xi}{\partial y}+\frac{\partial L_{mn}}{\partial\eta}\frac{\partial\eta}{\partial y}\right)\left(\frac{\partial u^e}{\partial\xi}\frac{\partial\xi}{\partial y}+\frac{u^e}{\partial\eta}\frac{\partial\eta}{\partial y}\right)|J^e|\,d\xi d\eta\end{aligned}$$

It is then a matter of rearranging the different terms.

$$\begin{aligned}\int_{\Omega_e} \nabla\left(l_m(\xi^e(x,y))l_n(\eta^e(x,y)\right)\cdot\nabla u^e\,dxdy &= F^{\xi\xi}_{mn;e}+F^{\xi\eta}_{mn;e}+F^{\eta\xi}_{mn;e}+F^{\eta\eta}_{mn;e} \\ F^{\xi\xi}_{mn;e} &= \int_{\hat{\Omega}_e}\frac{\partial L_{mn}}{\partial\xi}\frac{\partial u^e}{\partial\xi}\left(\frac{\partial\xi}{\partial x}\frac{\partial\xi}{\partial x}+\frac{\partial\xi}{\partial y}\frac{\partial\xi}{\partial y}\right)|J^e|d\xi d\eta \\ F^{\xi\eta}_{mn;e} &= \int_{\hat{\Omega}_e}\frac{\partial L_{mn}}{\partial\xi}\frac{\partial u^e}{\partial\eta}\left(\frac{\partial\xi}{\partial x}\frac{\partial\eta}{\partial x}+\frac{\partial\xi}{\partial y}\frac{\partial\eta}{\partial y}\right)|J^e|d\xi d\eta \\ F^{\eta\xi}_{mn;e} &= \int_{\hat{\Omega}_e}\frac{\partial L_{mn}}{\partial\eta}\frac{\partial u^e}{\partial\xi}\left(\frac{\partial\eta}{\partial x}\frac{\partial\xi}{\partial x}+\frac{\partial\eta}{\partial y}\frac{\partial\xi}{\partial y}\right)|J^e|d\xi d\eta \\ F^{\eta\eta}_{mn;e} &= \int_{\hat{\Omega}_e}\frac{\partial L_{mn}}{\partial\eta}\frac{\partial u^e}{\partial\eta}\left(\frac{\partial\eta}{\partial x}\frac{\partial\eta}{\partial x}+\frac{\partial\eta}{\partial y}\frac{\partial\eta}{\partial y}\right)|J^e|d\xi d\eta\end{aligned}$$

We will again use the GLL quadrature to compute those integrals. Once again, the fact that the quadrature nodes are the same as our local nodes will allow us to compute the different terms efficiently. Let us introduce a few definitions :

$$\begin{aligned}W^{\xi\xi}_{ij;e} &= w_iw_j|J^e(\xi_i,\eta_j)|\left(\frac{\partial\xi}{\partial x}\frac{\partial\xi}{\partial x}+\frac{\partial\xi}{\partial y}\frac{\partial\xi}{\partial y}\right)(\xi_i,\eta_j) \\ W^{\xi\eta}_{ij;e} &= w_iw_j|J^e(\xi_i,\eta_j)|\left(\frac{\partial\xi}{\partial x}\frac{\partial\eta}{\partial x}+\frac{\partial\xi}{\partial y}\frac{\partial\eta}{\partial y}\right)(\xi_i,\eta_j) \\ W^{\eta\eta}_{ij;e} &= w_iw_j|J^e(\xi_i,\eta_j)|\left(\frac{\partial\eta}{\partial x}\frac{\partial\eta}{\partial x}+\frac{\partial\eta}{\partial y}\frac{\partial\eta}{\partial y}\right)(\xi_i,\eta_j)\end{aligned}$$

The derivatives that appear in the expression above do not need an analytic expression for the inverse mapping to be available. We can compute them using the inverse of the jacobian matrix (ref here).

Let us also remind the definition of the derivation matrix H given by equation 1.12. Using the quadrature, we have :

$$\begin{aligned}F^{\xi\xi}_{mn;e} &= \sum_{a=0}^{p}\sum_{b=0}^{p}W^{\xi\xi}_{ab;e}l'_m(\xi_a)l_n(\eta_b)\left(\sum_{c=0}^{p}\sum_{d=0}^{p}u^e_{cd}l'_c(\xi_a)l_d(\eta_b)\right) \\ &= \sum_{a=0}^{p}\sum_{b=0}^{p}W^{\xi\xi}_{ab;e}H_{ma}\delta_{nb}\left(\sum_{c=0}^{p}\sum_{d=0}^{p}u^e_{cd}H_{ca}\delta_{db}\right) \\ &= \sum_{a=0}^{p}W^{\xi\xi}_{an;e}H_{ma}\left(\sum_{c=0}^{p}u^e_{cn}H_{ca}\right)\end{aligned}$$

If we denote U^e a matrix such that $(U^e)_{ij} = u^e_{ij}$ and $W^{\xi\xi}_e$ a matrix such that $(W^{\xi\xi}_e)_{ij} = W^{\xi\xi}_{ij;e}$ and if we denote the Hadamard product by $\circ$, then we can write :

$$F^{\xi\xi}_{mn;e} = \left(H(W^{\xi\xi}_e \circ H^T U^e)\right)_{mn}$$

For similar definitions, it can be showed that the order terms are given by :

$$\begin{aligned} F^{\xi\eta}_{mn;e} &= \left(H(W^{\xi\eta}_e \circ U^e H)\right)_{mn} \\ F^{\eta\xi}_{mn;e} &= \left((W^{\eta\xi}_e \circ H^T U^e)H^T\right)_{mn} \\ F^{\eta\eta}_{mn;e} &= \left((W^{\eta\eta}_e \circ U^e H)H^T\right)_{mn} \end{aligned}$$

We can see that the problem reduce to computing a few matrix products. The only thing left to do is to take hanging nodes into account when we gather the results :

$$(Au)^e_I = \sum_{m=0}^{p}\sum_{n=0}^{p} R^e_{mn}(I)\left(F^{\xi\xi}_{mn;e} + F^{\xi\eta}_{mn;e} + F^{\eta\xi}_{mn;e} + F^{\eta\eta}_{mn;e}\right)$$

And, of course, we need to take the influence of every quadrant into account :

$$(Au)_I = \sum_e (Au)^e_I$$

1.3 Preconditioned conjugate gradients

Now that we have our right hand side vector b and that we are able to compute a matrix-vector product Au, we can explain how to solve the linear system.

Since the number of degrees of freedom is huge, it is impossible to use a direct method to solve the linear system. Instead, we will turn ourselves to iterative methods.

The matrix A arises from the discretization of an Poisson equation using finite elements and therefore it is sparse, symmetric and positive-definite (see ref here). The conjugate gradients method is therefore well suited to solve the linear system. We have to note however that the convergence speed of the conjugate gradients depends on the condition number of the matrix A, $\kappa(A)$. Indeed, as given in ref here, if we denote $e_i = u - u_i$ where u is the solution of the linear system and u_i is our approximation after the i-th iteration, then :

$$||e_i||_A \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^i ||e_0||_A \tag{1.15}$$

Where $||e_i||_A = \left(e_i^T A e_i\right)^{\frac{1}{2}}$. We can see in equation 1.15 that the higher the condition number of A, the slower the convergence.

Unfortunately, the condition number of the matrix A is often very large and that is why we need to use a preconditioner.

Formally, we want to solve :

$$Au = b$$

Which is equivalent to solving :

$$M^{-1}Au = M^{-1}b$$

If M is easy to invert and $\kappa(M^{-1}A) \ll \kappa(A)$, then we should have a faster convergence at not too great a cost. The problem is that it is not possible to guarantee that $M^{-1}A$ is either symmetric nor positive definite, which is required for the CG.

However, we know that if M is symmetric and positive-definite, then there exists a matrix E such that $M = EE^T$. We can also note that $M^{-1}A$ and $E^{-1}AE^{-T}$ have the same eigenvalues. Indeed, let us assume that v if an eigenvector of $M^{-1}A$ associated with eigenvalue λ. Then :

$$E^{-1}AE^{-T}(E^Tv) = (E^TE^{-T})E^{-1}Av = E^TM^{-1}Av = \lambda E^Tv$$

And we can conclude that E^Tv is an eigenvector of $E^{-1}AE^T$ with eigenvalue λ. If the two matrices have the same eigenvalues, they also have the same condition number. If A is symmetric and positive-definite, it is clear that $E^{-1}AE^{-T}$ is also symmetric and positive-definite. We can thus use the CG to solve the following system (which is equivalent to $Au = b$) :

$$E^{-1}AE^{-T}\hat{u} = E^{-1}b \qquad\qquad \hat{u} = E^Tu$$

This formulation has the unwanted property of making explicit use of the matrix E. We would like to remove it entirely and use M only. It is indeed possible (see ref here).

Let us define u_i the approximation after the i-th approximation and $r_i = b - Au_i$. Then, the PCG procedure is given by algorithm 1. We can note that we indeed do not mention E but use M^{-1} instead. We can also see that we start with a zero initial guess. It does not have to be the case and if we had a better initial guess, we should use it.

Algorithm 1 Preconditioned Conjugate Gradients

$u_0 = 0$
$r_0 = b - Au_0$
$z_0 = M^{-1}r_0$
$p_0 = z_0$
$i = 0$
while $||r_i||_2 > \epsilon||r_0||_2$ **do**
  $d_i = Ap_i$
  $\alpha_i = \frac{r_i^Tz_i}{p_i^Td_i}$
  $u_{i+1} = u_i + \alpha_ip_i$
  $r_{i+1} = r_i - \alpha_id_i$
  $z_{i+1} = M^{-1}r_{i+1}$
  $\beta_{i+1} = \frac{z_{i+1}^Tr_{i+1}}{z_i^Tr_i}$
  $p_{i+1} = z_{i+1} + \beta_{i+1}p_i$
  $i = i + 1$

The stopping criterion is defined using the norm of the residual. We want it to be less than a given fraction of the norm of the initial residual.

The most important part in the algorithm presented is the preconditioner M^{-1}. Indeed, the convergence speed is a function of $\kappa(M^{-1}A)$. Therefore, we want a preconditioner that is easy to compute and such that the condition number of $M^{-1}A$ is a lot smaller than the one of A. Oftentimes, those two goals are contradictory.

As explained in the introduction, our preconditioner here consists of two parts. We have one coarse grid correction based on solving the problem on the mesh with an interpolation of degree $p = 1$. The resolution of this coarse problem will be done using a multigrid method. Let us call this part P^c. The second part is the fine preconditioner where we solve the problem exactly on overlapping subdomains (this is often called an additive Schwarz preconditioner). Let us call this part P^f. We then add the two part of the preconditioner :

$$M^{-1} = P^c + P^f$$

The hope is that the two parts of the preconditioner act on different parts of the problem and that the sum of the two makes a good preconditioner. The rest of this chapter is entirely about how to compute $M^{-1}r = P^c r + P^f r$.

1.4 Coarse preconditioner : the multigrid solver

Let us first present the coarse part of the preconditioner. It consists of solving the problem on the mesh using an interpolation of degree $p = 1$ with a geometric multigrid method.

Since in general we use an interpolation degree p, the first thing to do is to restrict this residual on the same mesh but for a degree $p = 1$. The first part of this section presents the restriction used. Similarly, once we have computed the solution for $p = 1$, we have to prolong it to a higher degree. To preserve the symmetry of the problem, we will define the prolongation operator to be the transpose of the restriction operator. The procedure explained here is an application with hanging nodes of the one described in ref here. Formally, if we denote by S the restriction operator and by G the geometric multigrid solver, we have :

$$P^c r = S^T G S r$$

The second part explains in more details what are the key components of the geometric multigrid solver. The practical implementation and the way it works with p4est is however described in the next chapter.

1.4.1 Restriction of r from a high degree to $p = 1$

Let us now define the operator S that will restrict the high degree residual r to a low degree $p = 1$. Authors of ref here have shown that a good choice for the restriction is the L^2 projection.

Let us call $V_p^h \in H^1(\Omega)$ the space whose basis consists of the high degree global basis functions (later denoted ϕ_i for $i = 1, ..., N$) and $V_1^h \in H^1(\Omega)$ the one whose basis is constituted by the bilinear global basis function (later denoted Φ_i for $i = 1, ..., M$). We want to find $R \in V_1^h(\Omega)$ such that R is the L^2 projection of $r \in V^p(\Omega)$ onto $V_1^h(\Omega)$. As shown in ref here, the L^2 projection is then given by :

$$R = C m^{-1} r$$

Where m is the high degree mass matrix and C is called the correlation matrix. The mass matrix m is easy to invert since in the case of spectral elements, it is diagonal, i.e. we have :

$$m_{ij} = \int_\Omega \phi_i \phi_j \, dxdy \approx 0 \qquad \text{if } i \neq j$$

The other term is the correlation matrix. It is defined by :

$$C_{I,i} = \int_\Omega \Phi_I \phi_i \, dxdy$$

We can also define its local counterpart using local shape functions. Let us denote l_i the one dimensional shape function of degree p associated with the i-th GLL node and L_i the one dimensional linear shape function associated with the i-th GLL node. Then we have :

$$C_{IJ,ij;e} = \int_{\Omega_e} L_I L_J l_i l_j \, dxdy$$

If we use the GLL quadrature using the $(p+1)^2$ GLL nodes on quadrant e, we get :

$$C_{IJ,ij;e} \approx w_i w_j |J^e_{ij}| \Phi_I(\xi_i)\Phi_J(\xi_j)$$

For later purposes, let us define :

$$m_{ij;e} = w_i w_j |J^e ij|$$
$$B_{IJ,ij} = \Phi_I(\xi_i)\Phi_J(\xi_j)$$

We can then compute the restriction of r as :

$$y = m^{-1} r$$
$$R = Cy$$

Applying m^{-1} to r is rather easy : we only scale the residual by the inverted mass matrix. We now have to transfer y_i for $i = 1, ..., N$ onto each local quadrant e. For this, let us use the operator R^e_{ij} to handle hanging nodes :

$$y_{ij;e} = \sum_{K=1}^{N} R^e_{ij}(K) y_K$$

Let us then compute the local coarse grid residual using the correlation matrix in its local form. This yields :

$$R_{IJ;e} = \sum_{i=0}^{p} \sum_{j=0}^{p} B_{IJ,ij} m_{ij;e} y_{ij;e}$$

The last remaining thing to do is to gather the local coarse grid residual to obtain the global coarse grid residual. Let us do not forget to handle the hanging nodes and therefore to use the operator R^e_{IJ} (not to confuse with $R_{IJ;e}$ the local coarse grid residual on quadrant e). Let us note that it is not the same as R^e_{ij} since it works on the global nodes for bilinear interpolation where R^e_{ij} is used for the interpolation of degree p.

$$R_K = \sum_e \sum_{I=0}^{1} \sum_{J=0}^{1} R^e_{IJ}(K) R_{IJ;e}$$

This residual R is then used as a right-hand side for the multigrid solver.

1.4.2 Prolongation of the solution from $p = 1$ to a high degree

Let us assume that the solution given by the multigrid solver with R as right-hand side is given by Z, i.e. :

$$Z = GR$$

We now need to prolong this coarse scale correction onto the fine grid. Since the prolongation operator is defined as the transpose of the restriction operator, we have :

$$P^c r = m^{-1} C^T Z$$

The global coarse correction Z is first scattered to each element to have the local coarse scale correction. Let us not forget to handle hanging nodes :

$$Z_{IJ;e} = \sum_{K=1}^{M} R^e_{IJ}(K) Z_K$$

Then, let us prolong the correction to obtain :

$$z_{ij;e} = \sum_{I=0}^{1} \sum_{J=0}^{1} B_{IJ,ij} m_{ij;e} Z_{IJ;e}$$

We then have to gather $z_{ij;e}$ to compute its global counterpart. Once again, this is where we handle hanging nodes.

$$z_K = \sum_{e} \sum_{i=0}^{p} \sum_{j=0}^{p} R^e_{ij}(K) z_{ij;e}$$

The last thing left to do is to scale z_K by the inverted mass matrix. This yields :

$$(P^c r)_K = \frac{z_K}{m_K}$$

1.4.3 The geometric multigrid solver

This subsection generally describe how the geometric multigrid method works and why it is so effective. However, a lot of details have been omitted and more information can be obtained in the relevant literature ref here.

Motivation

The first observation we have to make is that for any known function v such that $v(x, y) = u_0$ if $(x, y) \in \Gamma$, we can rewrite problem 1.1. Indeed, if we define a function $e = u - v$, then we have

$$\nabla^2 e = f - \nabla^2 v = r \qquad \text{on } \Omega \tag{1.16}$$

$$e = u - v = 0 \qquad \text{on } \Gamma \tag{1.17}$$

Which is qualitatively the same problem. But if we are able to solve it for e more easily than we would have been able to do it for u, then we can simply recover u from e :

$$u = v + e$$

The second observation we have to make is that the discretization of problem 1.1 or equivalently the one above yields a linear system of equations to solve (as the one derived in the first section of this chapter). As already mentioned, the matrix A defining this linear system has very often a very large condition number $\kappa(A)$. As a result, using iterative methods can be very slow. Typically, the condition number of A is $\mathcal{O}(h^2)$ where h is the size of the smallest quadrant ref here. Thus, if we use a coarser grid (where the smallest quadrant has a larger size than before), the problem is easier to solve.

The idea behind multigrid is therefore a two-level idea. It works as follow :

1. Given an approximate solution to the problem, compute the residual $f - \nabla^2 v$.
2. Solve problem 1.16 to obtain e. It should be easier to solve than the initial problem by using a coarser grid.
3. Add e to v to get the solution u

The next step in the reasoning is to use this idea recursively. Indeed, in step 2, to solve problem 1.16, one can reuse the same process. A more precise definition of the algorithm is given at the end of this section, in algorithm number here.

We still have not explained two things in the procedure above. The first is how to choose an approximate solution to the problem. In theory, it can be any but we have to remember that, in step 2, we will use a coarser grid to solve for e. As a result, we want an approximation such that $e = u - v$ is a smooth function that a coarse grid can capture well. One way to obtain this approximation is by using an iterative method. Indeed, it has been shown (ref here) that even tough iterative methods like Jacobi or Gauss-Seidel have a tendency to stall, just a few iterations allows them to capture the high-frequency modes in the solution. That means that if we compute v by doing some iterations of the Jacobi method, then $e = u - v$ will contain only low frequency modes. And a solution containing low frequencies can be computed on a coarser grid.

The second thing we have to consider is how to transfer one function onto another grid. Indeed, in step 1, we compute the residual $f - \nabla^2 v$ but then we have to solve problem 1.16 on a different, coarser grid. We therefore need an operator to compute the right-hand side of the linear system on the coarser grid from the right-hand side of the one on the initial grid. This is called the restriction operator. Similarly, when we have computed the solution on the coarser grid in step 2, we need to update our approximation v with e on the initial grid. This operator is called the prolongation operator. We will see that there exists a link between those two operators.

The smoother : Jacobi

The next step in the reasoning is to use this idea recursively. Indeed, in step 2, to solve problem 1.16, one can reuse the same process. A more precise definition of the algorithm is given at the end of this section, in algorithm number here.

We still have not explained two things in the procedure above. The first is how to choose an approximate solution to the problem. In theory, it can be any but we have to remember that, in step 2, we will use a coarser grid to solve for e. As a result, we want an approximation such that $e = u - v$ is a smooth function that a coarse grid can capture well. One way to obtain this approximation is by using an iterative method. Indeed, it has been shown (ref here) that even tough iterative methods like Jacobi or Gauss-Seidel have a tendency to stall, just a few iterations allows them to capture the high-frequency modes in the solution. That means that if we compute v by doing some iterations of the Jacobi method, then $e = u - v$ will contain only low frequency modes. And a solution containing low frequencies can be computed on a coarser grid.

The second thing we have to consider is how to transfer one function onto another grid. Indeed, in step 1, we compute the residual $f - \nabla^2 v$ but then we have to solve problem 1.16 on a different, coarser grid. We therefore need an operator to compute the right-hand side of the linear system on the coarser grid from the right-hand side of the one on the initial grid. This is called the restriction operator. Similarly, when we have computed the solution on the coarser grid in step 2, we need to update our approximation v with e on the initial grid. This operator is called the prolongation operator. We will see that there exists a link between those two operators.

Chapter 2

Results and discussion

In this chapter, we will put the algorithms presented in the theory chapter to the test and present the results. The implementation of the algorithms consists of a code written in C of more than 8000 lines which leverages the p4est library (see the implementation chapter for information) for the mesh generation and refinement as well as the Lapack library (a widely used linear algebra library, see [1]) to solve linear systems and diagonalize matrices. We will look at the experimental results and compare them to the theory developed. The code written can in principle handle any order of interpolation and any geometry but in practice most tests have been performed for $p = 2, 4, 6, 8$ and on $\Omega = [-1; 1]^2$ which allowed us to have analytic solutions to the problems we investigated.

This chapter is divided into several sections. The first looks at the geometric multigrid preconditioner. As explained in the theory chapter, it cannot be used as the only preconditioner since it has a kernel but we can use it as an iterative solver for $p = 1$. This is done to verify an important property of the geometric multigrid methods : the h-independent convergence. In this section, we also look at the influence of hanging nodes on the solver.

The second section focus on the preconditioned conjugate gradients with only the fine preconditioner. This aims at looking at the properties of the overlapping Schwarz preconditioner presented in the theory chapter. We will first test it on regular elements, where the preconditioner is optimal, then look at what happens when we have a mesh with distorted elements. We will afterwards move on to non conforming meshes and look at the influence of having hanging nodes on the fine preconditioner. Finally, we will increase the degree of the interpolation and observe how the number of iterations of PCG evolves.

The third section looks at the PCG with the two scale preconditioner : both the coarse grid correction computed by the multigrid solver and the fine scale correction computed by the overlapping Schwarz method. The addition of the coarse scale preconditioner should provide a convergence independent of the number of quadrants. As before, we will first test the algorithms on a regular mesh, where it should perform very well. Then, we will see how it fares on meshes with distorted elements and what happens when we increase the distortion. We will thereafter consider non conforming meshes obtained through adaptive mesh refinement. We will check that here also we have a convergence that is independent of the number of quadrants and how the number of iterations of PCG compares to the case of conforming meshes. Then, we increase the degree of the interpolation on conforming meshes and look at what happens to the number of iterations. Finally, we emphasize the importance of being able to handle higher orders of interpolation. Given a wanted accuracy on the L^2-norm of the error, we present the best polynomial order to use for a given problem and a given mesh.

The chapter ends on a partial conclusion which summarize the results presented before.

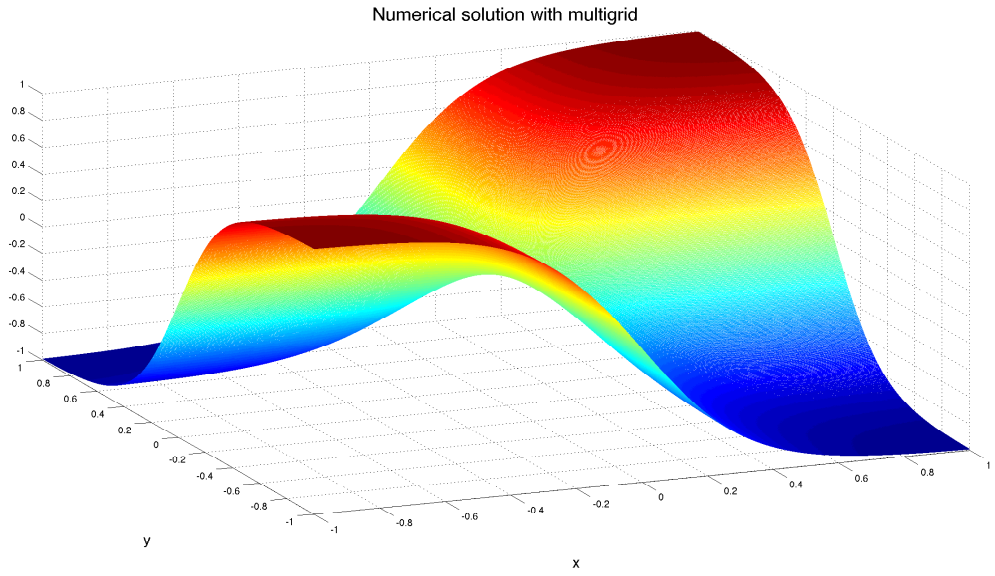


Figure 2.1: Numerical solution using the multigrid solver of $\nabla^2 u = f$ for $f = -2\tanh(3x)\tanh(3y)(18 - 9\tanh^2(3x) - 9\tanh^2(3y))$

2.1 Multigrid

In this section, we will test the coarse part of the preconditioner : the multigrid solver. This will be done in two steps. First, we will verify a well known property of the multigrid solvers : the h-independent convergence. We will also compare the number and iterations needed while varying key parameters of the model. Those tests will be performed on various meshes. The second part will focus on the influence of hanging nodes on the numerical solution.

Let us before all present a type of numerical solution that can be obtained using the multigrid solver. Figure 2.1 shows an example of the numerical solution computed. We can see that even with $p = 1$, we have a good approximation. This is because the forcing term is not at all oscillatory.

2.1.1 H-independent convergence

Let us first verify that our geometric multigrid solver has the required property and that the same number of iterations is needed to obtain a given accuracy, however small the elements. We will use the model problem throughout this section with the same right hand side. For all the tests below, the domain will be : $\Omega = [-1; 1]^2$. We will solve :

$$\nabla^2 u = -\frac{\pi^2}{2}\cos(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y) \qquad \text{on } \Omega \tag{2.1}$$

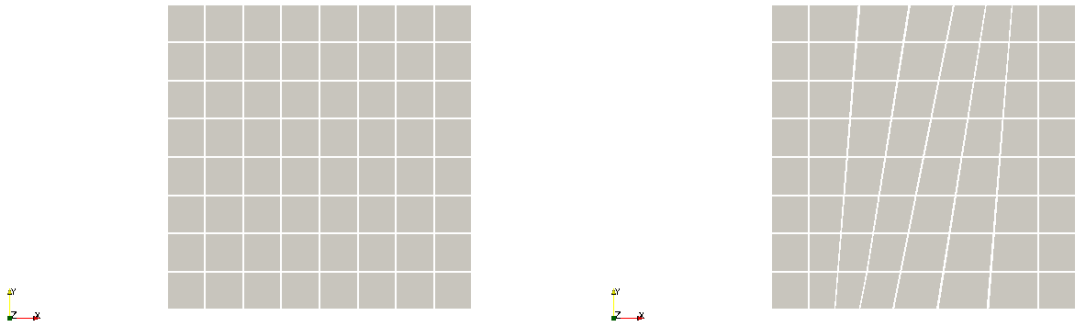
$$u = 0 \qquad \text{on } \Gamma \tag{2.2}$$

It is easy to see that for the given domain, we have an analytic solution :

$$u(x, y) = \cos(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y))$$

Let us now explain how we define the error. We will look at the absolute difference between the value of the approximation and the value of the analytic solution at the global nodes and take the maximum. Formally, we have that the error after iteration k, e_k is :

$$e_k = \max_i |u(x_i, y_i) - u_i^k|$$



(a) Regular mesh

(b) Crooked mesh

Figure 2.2: The two "supra meshes" that will be refined during the tests for the multigrid solver. We have one regular mesh (left) where the elements are squares and one crooked mesh (right) where the elements are slightly distorted.

Where u_i^k is the value of our approximation at the global node i after iteration k. Since $u_i^0 = 0$ for all i, it is clear that $e_0 = 1$.

Figure 2.2 shows the two "supra meshes" that we will refine during the tests. Some refinements will be uniform and some will be so that we have the presence of hanging nodes. We can note that even for the crooked mesh, the elements are not really distorted. It does not matter since we are only testing the h-independent convergence. Having a mesh with elements that are more distorted will only influence the accuracy of the approximation and not how the algorithm solve the linear system we want to solve.

Let us start with a simple V-cycle on the regular mesh (figure 2.2a) that we will refine uniformly. We will compare the errors when we increase the number of degrees of freedom and for different ν_1 and ν_2. The sum of the two smoothing parameters is chosen to be constant so that we have the same number of Jacobi iterations for all pairs ν_1 and ν_2. Here, we chose the sum to be equal to four.

The results are shown in table 2.1. We can see that we indeed have an h-independent convergence of the solver. For every pair of the smoothing parameters, at least for the first few iterations, the error e_k is identical for all values of N. Except for the pair $\nu_1 = 4$ and $\nu_2 = 0$, each iteration roughly decrease the error by one decimal point.

We can also note that the values of the parameters influence the convergence. For this particular problem and this particular mesh, the values $\nu_1 = 0$ and $\nu_2 = 4$ seem to be the best as the error is smaller after the same number of iterations than for the other pairs. The less post smoothing iterations (ν_2) we do, the slower the convergence. This is true for $\nu_2 = 1$ where the error on the finest mesh after six iterations is a hundred times larger than on the same mesh after the same number of iterations for $\nu_2 = 4$, and it is clearer still for $\nu_2 = 0$ where the error is a thousand times larger after six iterations on the finest mesh.

We have to note that even tough the errors are identical for all meshes at first, we have a difference after a few iterations. This can be explained by the fact that our geometric multigrid algorithm actually solves a linear system whereas the error is measured as the difference between the analytic solution and the solution of the linear system. Thus, even if we solved the linear system exactly, we would still have an error and that error should decrease as the number of degrees of freedom increases. This is indeed what we observe here. For example, for the mesh with $N = 2.6\,10^5$, we can see that after six iterations, we have almost converged and that the error stays around $3.14\,10^{-6}$. Even if we did several more iterations, the error would not decrease significantly. That is because we have the solution of the linear system and the error is only due

N	$2.6\ 10^5$	$1.1\ 10^6$	$4.2\ 10^6$	$1.7\ 10^7$	$6.7\ 10^7$
	$\nu_1 = 2$	$\nu_2 = 2$			
e_1	3.56e-02	3.56e-02	3.56e-02	3.56e-02	3.56e-02
e_2	1.34e-03	1.34e-03	1.34e-03	1.34e-03	1.34e-03
e_3	5.42e-05	5.66e-05	5.72e-05	5.73e-05	5.73e-05
e_4	3.11e-06	2.90e-06	3.45e-06	3.59e-06	3.62e-06
e_5	3.30e-06	9.48e-07	3.62e-07	3.50e-07	3.85e-07
e_6	3.17e-06	8.14e-07	2.26e-07	8.26e-08	5.23e-08
	$\nu_1 = 3$	$\nu_2 = 1$			
e_1	3.70e-02	3.71e-02	3.71e-02	3.71e-02	3.71e-02
e_2	1.62e-03	1.63e-03	1.63e-03	1.63e-03	1.63e-03
e_3	1.04e-04	1.06e-04	1.07e-04	1.07e-04	1.07e-04
e_4	1.10e-05	1.22e-05	1.27e-05	1.29e-05	1.29e-05
e_5	4.47e-06	2.20e-06	1.96e-06	2.10e-06	2.13e-06
e_6	3.34e-06	1.00e-06	4.38e-07	3.53e-07	3.88e-07
	$\nu_1 = 1$	$\nu_2 = 3$			
e_1	3.57e-02	3.57e-02	3.57e-02	3.57e-02	3.57e-02
e_2	1.29e-03	1.29e-03	1.29e-03	1.29e-03	1.29e-03
e_3	4.66e-05	4.89e-05	4.95e-05	4.96e-05	4.97e-05
e_4	1.53e-06	1.53e-06	2.10e-06	2.24e-06	2.28e-06
e_5	3.08e-06	7.31e-07	1.43e-07	1.17e-07	1.51e-07
e_6	3.14e-06	7.83e-07	1.95e-07	4.80e-08	1.13e-08
	$\nu_1 = 4$	$\nu_2 = 0$			
e_1	4.55e-02	4.55e-02	4.55e-02	4.55e-02	4.55e-02
e_2	3.26e-03	3.26e-03	3.26e-03	3.26e-03	3.26e-03
e_3	4.64e-04	4.71e-04	4.71e-04	4.71e-04	4.71e-04
e_4	9.24e-05	9.60e-05	9.65e-05	9.67e-05	9.67e-05
e_5	1.74e-05	2.04e-05	2.12e-05	2.14e-05	2.14e-05
e_6	6.17e-06	3.86e-06	4.55e-06	4.74e-06	4.78e-06
	$\nu_1 = 0$	$\nu_2 = 4$			
e_1	3.58e-02	3.58e-02	3.58e-02	3.58e-02	3.58e-02
e_2	1.29e-03	1.29e-03	1.29e-03	1.29e-03	1.29e-03
e_3	4.53e-05	4.77e-05	4.82e-05	4.84e-05	4.84e-05
e_4	1.17e-06	1.31e-06	1.89e-06	2.03e-06	2.07e-06
e_5	3.03e-06	6.80e-07	9.14e-08	7.66e-08	1.12e-07
e_6	3.13e-06	7.76e-07	1.87e-07	4.04e-08	3.62e-09

Table 2.1: Errors after k iterations of a V-cycle (e_k) for the regular mesh uniformly refined to have N degrees of freedom and for different values of the parameters ν_1 and ν_2

N	$2.6\,10^5$	$1.1\,10^6$	$4.2\,10^6$	$1.7\,10^7$	$6.7\,10^7$
	$\nu_1 = 2$	$\nu_2 = 2$			
e_1	3.64e-02	3.64e-02	3.64e-02	3.64e-02	3.64e-02
e_2	2.56e-03	2.49e-03	2.46e-03	2.44e-03	2.43e-03
e_3	7.41e-04	6.33e-04	5.80e-04	5.54e-04	5.41e-04
e_4	6.15e-04	3.15e-04	1.62e-04	1.35e-04	1.28e-04
e_5	5.96e-04	3.01e-04	1.52e-04	7.67e-05	4.76e-05
e_6	5.91e-04	2.98e-04	1.50e-04	7.50e-05	3.77e-05

Table 2.2: Errors after k iterations of a V-cycle (e_k) for the crooked mesh uniformly refined to have N degrees of freedom and for $\nu_1 = 2$ and $\nu_2 = 2$

Meshes	No hanging nodes	Figure 2.3a	Figure 2.3b
N	$4.2\,10^6$	$7.3\,10^6$	$7.0\,10^7$
e_1	3.56e-02	3.56e-02	3.56e-02
e_2	1.34e-03	1.34e-03	1.34e-03
e_3	5.72e-05	5.72e-05	5.74e-05
e_4	3.45e-06	3.47e-06	3.64e-06

Table 2.3: Errors after k iterations of a V-cycle (e_k) for a mesh without hanging nodes and the two meshes presented in figure 2.3. Each mesh has N degrees of freedom and the smoothing parameters were $\nu_1 = 2$ and $\nu_2 = 2$.

to the discretization. If we refine the mesh and go to $N = 6.7\,10^7$, then we can get smaller errors (of the order of 10^{-8}).

Let us now explore the results for the crooked mesh (figure 2.2b). Here also, we should expect an h-independent convergence. We only show the results for $\nu_1 = 2$ and $\nu_2 = 2$ but the same commentary applies for the other pairs. The results can be seen on table 2.2

We can see that the for the first few iterations, the error e_k is independent of the mesh. However, the note we made earlier is much clearer here. Because the mesh is not regular, the effect of discretization are more important and therefore we will not reach the same accuracy than we did before. That is why after six iterations, the less refined grid ($N = 2.6\,10^5$) still has an error of $5.91\,10^{-4}$. More iterations will not have a great impact on the solution since the error is mostly due to the discretization.

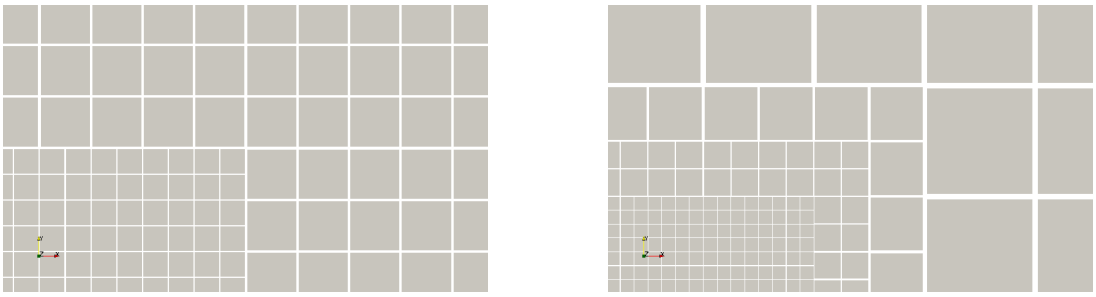
2.1.2 Influence of hanging nodes

We will now investigate the influence of hanging nodes on our solution. We will present the results only for the regular mesh but the tests have been performed on both and the same conclusions apply to the crooked mesh.

We will compare the values of the error between one mesh with no hanging nodes, one where we only have refined the lower left part of the domain once, and one where we have a rapid transition between two parts of different refinement level (which will occur often in AMR).

Figure 2.3 presents the latter two meshes. For the mesh in figure 2.3b, we have refined thrice more in a certain region than in the adjacent one. Since we do not allow adjacent quadrants to be more than one level apart, we obtain a "layer" where the levels of the quadrants is rapidly changing. In order to compare solutions, we made sure that the largest quadrants in all three meshes had the same size. This means that the meshes with hanging nodes have a lot more degrees of freedom.

Table 2.3 shows the results for the three different meshes using a V-cycle and with the smoothing parameters $\nu_1 = 2$ and $\nu_2 = 2$. Here again, we can see that the convergence is h-independent and that one iteration gives us roughly one more decimal. The presence of hanging



(a) Two levels (b) Levels rapidly changing

Figure 2.3: Zoom on a certain part of the two meshes containing hanging nodes that will be used to test the multigrid solver. We have one mesh where we only have refined a part of the domain once more than the rest (left) and a mesh where we have refined a part thrice more than the rest which results in levels changing rapidly (right).

nodes has no influence on the error we observe after a given number of iterations. The same result is observed with all pairs of the smoothing parameters and with other meshes.

This will be important in the next sections when we will use our multigrid solver as a preconditioner. Indeed, we will see that it is the coarse correction that allow for h-independent convergence.

2.2 Fine preconditioner

Let us now move on to the fine part of the preconditioner : the overlapping additive Schwarz preconditioner. We will test it by using the preconditioned conjugate gradients method described earlier but, for now, the preconditioner will only consist of the fine part (i.e. $P = P^f$).

As in the previous section, we will perform the tests in two parts : first, we will use meshes with elements that are distorted or not but with no hanging nodes. Then, we will see how the fine preconditioner performs in the presence of hanging nodes also for meshes that are distorted or not. Here, of course, we will use interpolations of higher degree. Typically, the tests will be performed for $p = 2, 4, 6, 8$.

2.2.1 No hanging nodes

Let us first present the problem we will use throughout this section. The forcing term will be chosen more oscillatory than in the previous part since we use interpolations of higher degree. As before, the domain is : $\Omega = [-1; 1]^2$ and Γ is the boundary. The problem is :

$$\nabla^2 u = -8\pi^2 \sin(2\pi x) \sin(2\pi y) \qquad \text{on } \Omega \tag{2.3}$$

$$u = 0 \qquad \text{on } \Gamma \tag{2.4}$$

This problem has an analytic solution and it is easy to convince oneself that this solution is given by :

$$u(x, y) = \sin(2\pi x) \sin(2\pi y)$$

Figure 2.4 shows the numerical solution to the problem above for $p = 2$ and $1.0\,10^6$ degrees of freedom for a regular mesh. We can note that it is exactly the same number of degrees of freedom as if we had refined uniformly once more and used an interpolation of degree $p = 1$. Let us then compare how the two approximations perform. We solved the problem for $p = 1$ with

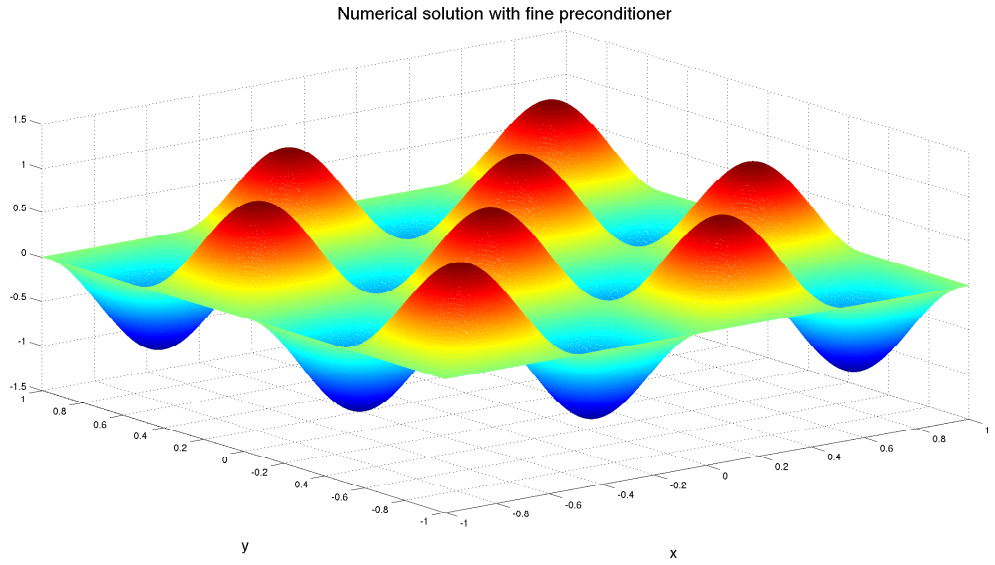


Figure 2.4: Numerical solution to problem 2.3 using an interpolation of order $p = 2$ and $1.0\,10^6$ degrees of freedom on a regular mesh with no hanging nodes.

our multigrid solver and the problem for $p = 2$ with the PCG and the fine preconditioner. Let us denote u_i^j as the value of the approximation for $p = j$ at node i. We have that :

$$e^1 = \max_i |u_i^1 - u(x_i, y_i)| = 5.02\,10^{-5}$$
$$e^2 = \max_i |u_i^2 - u(x_i, y_i)| = 1.01\,10^{-9}$$

We can see that with the same number of degrees of freedom, an approximation using $p = 2$ is much more accurate. This is because the solution is really smooth and is better approximated using a higher order interpolation than a bilinear interpolation on smaller quadrants. This is one example of the reasons we want to use higher order interpolations.

Regular meshes

Let us now move on to the comparison for different degrees of the number of iteration needed to reach a given accuracy as a function of the number of quadrants. We will take our regular mesh and uniformly refine it. Then, for $p = 2, 4, 6, 8$, we will see how many iterations are needed to reach a given error on the norm of the residual. Let us denote r_k the residual after iteration k of the preconditioned conjugate gradients. Of course, since our initial guess is zero, we have that $r_0 = b$ (since we are solving the linear system $Au = b$). For the following tests, our stopping criterion is given by :

$$\frac{||r_k||_2}{||r_0||_2} < 10^{-3}$$

Figure 2.5 shows the results. To put the data in perspective, we also have to show the number of degrees of freedom. Indeed, for a given number of quadrants, the higher degree the interpolation is, the more nodes we have. Table 2.4 contains the number of nodes for each mesh and for each degree p.

We can see that, even without the coarse preconditioner, we are solving the system in a small number of iterations compared to the number of degrees of freedom. For example, we only do

Number of quadrants	16^2	32^2	64^2	128^2	256^2
$p = 2$	$1.1\ 10^3$	$4.2\ 10^3$	$1.7\ 10^4$	$6.6\ 10^4$	$2.6\ 10^5$
$p = 4$	$4.2\ 10^3$	$1.7\ 10^4$	$6.6\ 10^4$	$2.6\ 10^5$	$1.1\ 10^6$
$p = 6$	$9.4\ 10^3$	$3.7\ 10^4$	$1.5\ 10^5$	$5.9\ 10^5$	$2.4\ 10^6$
$p = 8$	$1.7\ 10^4$	$6.6\ 10^4$	$2.6\ 10^5$	$1.1\ 10^6$	$4.2\ 10^6$

Table 2.4: Number of degrees of freedom for a regular mesh with different number of quadrants and for different degrees of interpolation.

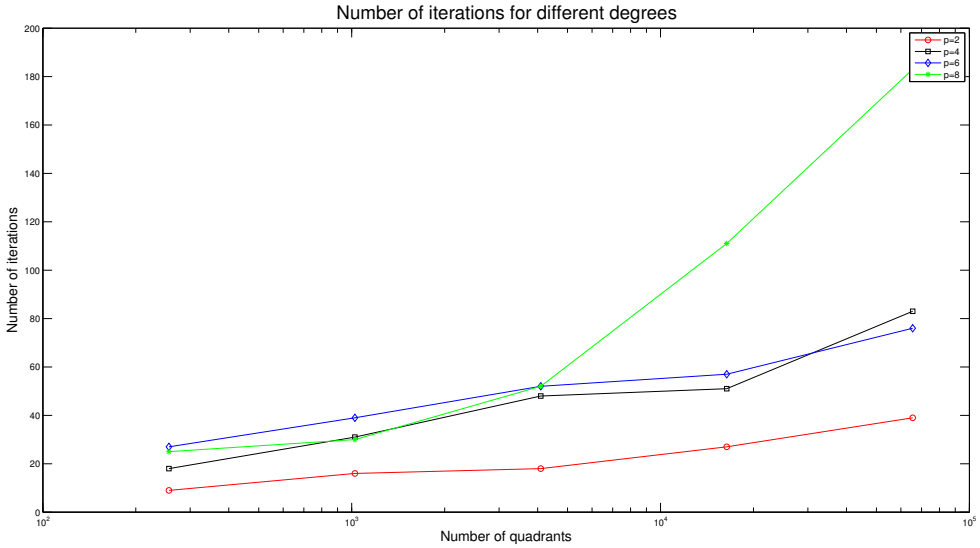
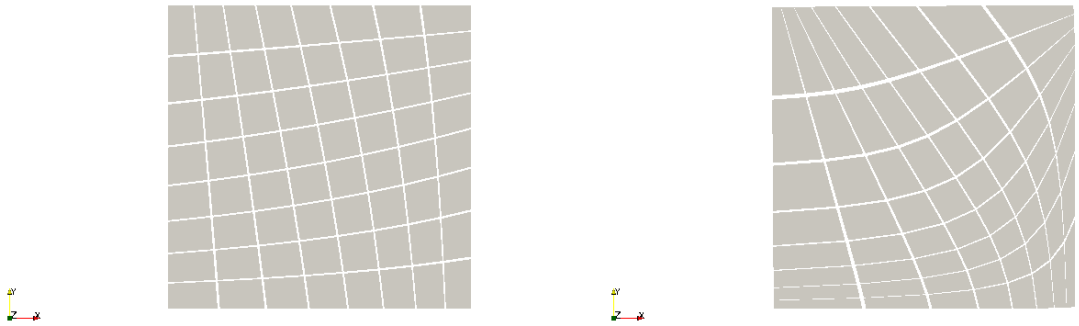


Figure 2.5: Number of iterations of PCG with only the fine preconditioner for different degree p of interpolation as a function of the number of quadrants in a regular mesh.



(a) Mesh with $a = 1.1$ (b) Mesh with $a = 1.4$

Figure 2.6: Examples of the meshes used for the tests of the fine preconditioner with distorted elements. The meshes are created using GMSH with its progression tool. The key parameter is the common ratio a that we will be increasing progressively.

about 80 iterations to solve the system with $2.4\,10^6$ degrees of freedom and with interpolations of degree $p = 6$.

We can also see that for every degree, the number of iterations increases when we refine the mesh. This is to be expected since the information from the boundaries has to go through more quadrant before propagate to the entire domain. Asymptotically, the number of iterations is expected to double as the number of quadrants is multiplied by four (i.e. the mesh size is divided by two). We can see that it is not yet the case here.

A last remark we can make is that the number of iterations tends to increase when the degree of the interpolation increases. This is especially true for the finest mesh where we need 183 iterations for $p = 8$ where we only need 39 iterations for $p = 2$. This can be explained by the fact that the size of the overlap decreases when p grows. As mentioned in [2], this issue would be fixed if we imposed a constant overlap.

Meshes with distorted elements

Let us now move on to meshes that are not regular anymore. Let us remember that when we developed the fine preconditioner, we assumed that the elements were rectangular which allowed us to compute the analytic solution to the problem. This part explores the influence of having distorted elements on the number of iterations needed to obtain a given accuracy.

To control the deformation, we will continually deform the regular mesh using the progression tool of GMSH (information about GMSH can be found in [3]). The deformation is performed using a geometric progression, using the common ratio a as the parameter. Obviously, the higher the parameter a, the more distorted the mesh is. It is clear that for $a = 1$, we have a regular mesh. Figure 2.6 presents three meshes of obtained with the progression with GMSH. On the left we used $a = 1.1$ and on the right we used $a = 1.4$.

Using the same tolerance as in the previous part, we ran the preconditioned conjugate gradients with the fine preconditioner for those new meshes. The order of the interpolation used is $p = 2$. The results are given in figure 2.7.

We can see, as it was expected, that the more we deform the mesh, the more iterations we need to do in order to obtain the wanted accuracy. The fact that, for distorted elements, we do not invert exactly the system but use an approximation allows us to compute the fine preconditioner efficiently even when we have a lot of elements but we can see here that it also costs more iterations of the preconditioned conjugate gradients. However, the gain is still huge.

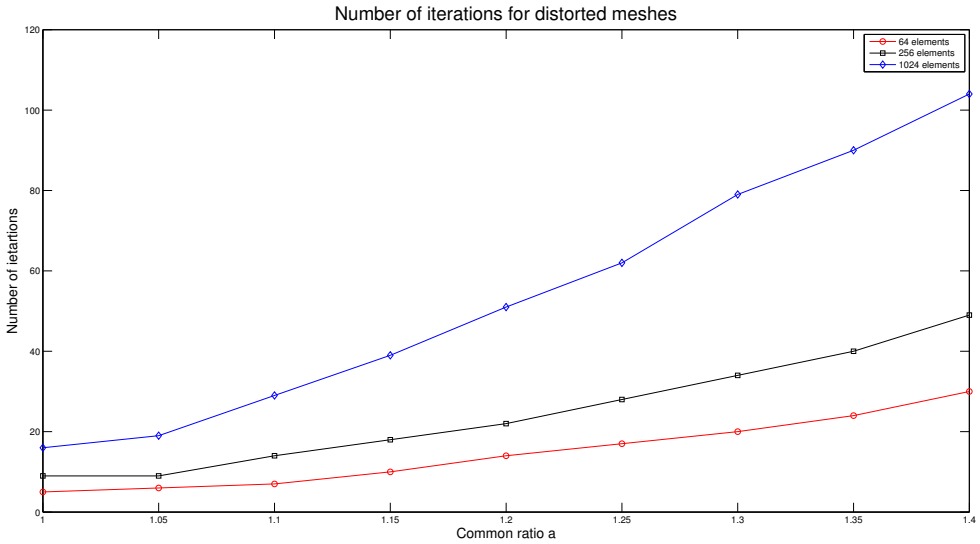


Figure 2.7: bla

Indeed, for a degree of interpolation p, we have $(p+3)^2$ nodes per overlapping subdomain. This means a complexity of $\mathcal{O}((p+3)^6)$ to solve the system exactly. Instead, with the method we use, the only need to do a few matrix multiplications where the matrices have $(p+3)$ rows and columns. This means a complexity of $\mathcal{O}((p+3)^2)$. Even with $p=2$, the results shows that it is much faster to not take geometric factors into account.

We can also note that the effect of increasing the number of elements (i.e. reducing the mesh size) on the number of iterations is clearer here. For example, for the mesh with $a = 1.15$, we need 10 iterations for 8^2 elements, 18 iterations for 16^2 elements and 39 iterations for 32^2 elements. The same explanation applies here : when we multiply the number of elements by four, we roughly multiply the number of quadrants in each direction by two and therefore the information needs twice as many iterations to propagate to the domain.

2.2.2 Influence of hanging nodes

In this part, we will explore the influence of hanging nodes on the number of iterations needed by the preconditioned conjugate gradients with the fine preconditioner to converge. Because we want meshes that are not artificial and come from real AMR applications, we will change the forcing term in this part and we will also use a recursive refine function when we build the mesh with p4est.

Let us first define the problem and the forcing term. As before, the domain is : $\Omega = [-1;1]^2$ and Γ is the boundary. We will solve :

$$\nabla^2 u = -2\tanh(nx)\tanh(my)\left[n^2(1-\tanh(nx)^2) + m^2(1-\tanh(my)^2)\right] \quad \text{on } \Omega \qquad (2.5)$$

$$u = \tanh(nx)\tanh(my) \quad \text{on } \Gamma \qquad (2.6)$$

We can see that problem 2.5 has an analytic solution that is given by :

$$u(x,y) = \tanh(nx)\tanh(my)$$

The parameters n and m can be adjusted to make the jump in the hyperbolic tangent more steep. An example of a numerical solution with $p=1$ and obtained by our multigrid solver has already been shown on figure 2.1 for $n=3$ and $m=3$.

Tol	0.020	0.015	0.010
Number of quadrants	1276	1384	3064
hang	14.05%	16.61%	22.13%

Table 2.5: Number of quadrants obtained using the recursive refine function on the problem 2.5 for different tolerances as well as the ratio *hang* for each mesh with an interpolation of degree $p = 2$.

As explained before, we will use a recursive refine function when we build the forest. Since we know the analytic solution, we can cheat a little and use it for the refinement process. We will ask that the absolute value of the difference between the value of u in the center of the quadrant and the mean of the values of u at the four corners of the quadrant is less than a fixed tolerance multiplied by the maximum value of the function. So, if the four corners have coordinates (x_i, y_i) for $i = 0, 1, 2, 3$, and that $u_{max} = max_{x,y\in\Omega} |u|$, we impose the following rule :

$$\left| u(\frac{1}{4}\sum_{i=0}^{3} x_i, \frac{1}{4}\sum_{i=0}^{3} y_i) - \frac{1}{4}\sum_{i=0}^{3} u(x_i, y_i) < u_{max} tol \right| \tag{2.7}$$

Where *tol* is a fixed tolerance. Intuitively, the tighter the tolerance, the more refined the grid needs to be and the more hanging nodes we will have thanks to the jump in the hyperbolic tangent function.

Increasing the relative number of hanging nodes

To have a rather steep jump, we chose $n = m = 12$ for the different tests that follow. We varied the tolerance to have more or less hanging nodes. We also needed a way to quantify the presence of hanging nodes. Let us define *hang*, the ratio of the number of hanging nodes over the number of global nodes.

$$hang = \frac{\#\text{hanging nodes}}{\#\text{global nodes}} \tag{2.8}$$

Table 2.5 shows this number for different values of the tolerance and for $p = 2$. We can see, as expected, that the ratio *hang* increases while we make the tolerance tighter. Of course, the exact value of *hang* does not matter since even for a given mesh, it will change with the degree of the interpolation so it is rather how it evolves that interests us.

Let us now see how the number of iterations varies for the different meshes. Figure 2.8 shows the number of iterations needed to reach the same norm on the residual as before for the three different meshes obtained with the tolerances presented in table 2.5. We can see on the graph that the number of iterations increases when we have relatively more hanging nodes. We have to be careful since this phenomenon can also be explained by the fact that we have more quadrants when the tolerance gets tighter and that also causes an increase in the number of iterations as showed earlier in this section. However, it is observed that we need less iterations when we do not have hanging nodes and for a similar number of quadrants (for example, for 1024 quadrants without hanging nodes, we need 18 iterations whereas we need 38 for 1276 quadrants).

As explained in the theory chapter, when we have hanging nodes, we do not treat all hanging possibilities and therefore we have an error when we compute the local residual. This explains why we need more iterations in presence of hanging nodes and the fact that we converge less quickly the more hanging nodes we have in the mesh. However, the number of iterations is still acceptable.

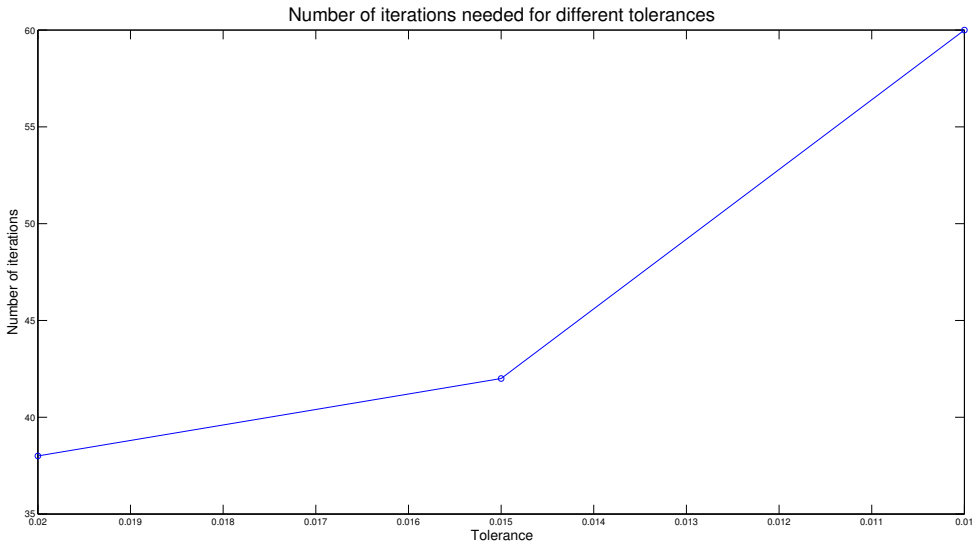


Figure 2.8: Number of iterations needed to reach a given norm on the residual for a degree of interpolation $p = 2$ and for meshes obtained with different tolerances.

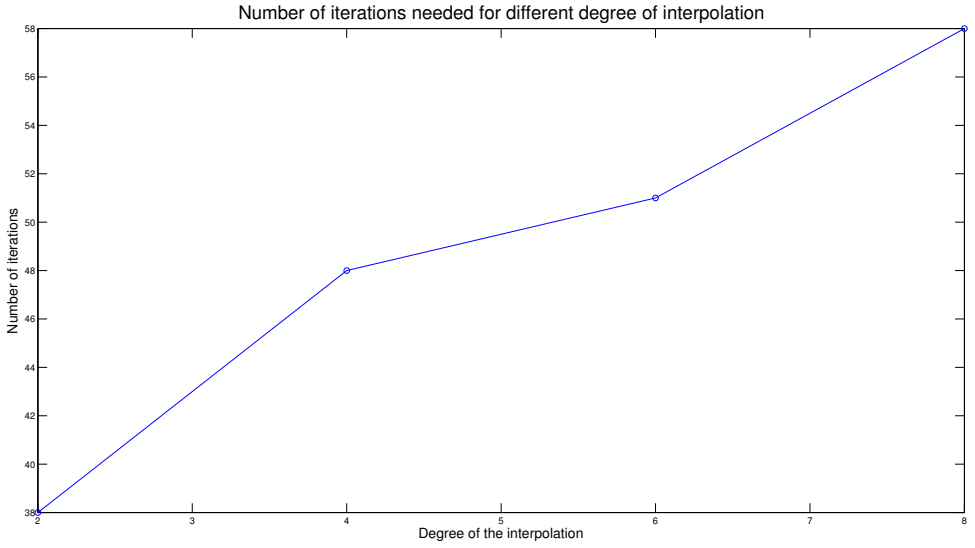


Figure 2.9: Number of iterations needed to reach the tolerance of the norm of the residual as a function of the degree of the interpolation used for a mesh obtained using $tol = 0.02$ in the recursive refine function.

Increasing the degree of the interpolation

Let us finally look at what happens when we increase the degree of the interpolation. We tried different degrees of interpolation ($p = 2, 4, 6, 8$) and look at the number of iterations needed to obtain the numerical solution. Figure 2.9 shows the results.

We can see that the number of iterations increases with the degree. We need 38 iterations for $p = 2$ but 58 for $p = 8$. As already explained in the case with no hanging nodes, this is in part due to the fact that when we increase the degree of the interpolation, the size of the overlap decreases and therefore we need more iterations.

We can also mention the fact that since we do not treat all hanging possibilities when we compute the local residual at the overlaps, increasing the degree (and therefore the number of nodes in the overlaps) might have a effect on the number of iterations.

All the tests presented here were on meshes where the elements were not distorted and where we expect the fine preconditioner to behave optimally but the same tests have been performed with distorted quadrants and we observe the same qualitative results.

2.3 Two scale preconditioner

Let us finally move on to the two-scale preconditioner. We showed in the previous part that using only the fine scale preconditioner was not a good idea when the number of quadrants increased (the number of iterations roughly doubles when the mesh size is divided by two). As explained in the theory chapter, the idea is to add a coarse part in the preconditioner (i.e. $P = P^f + P^c$), consisting mainly of solving a problem with an interpolation degree $p = 1$ with a geometric multigrid method, so that the number of iterations stays constant when we increase the number of quandrants.

As in the previous section, the tests will be performed in two parts : first, we will analyse the performances of the two-scale preconditioner when there are no hanging nodes (but for both regular and distorted meshes) and then we will look at what happens when the forest of quadtrees is refined recursively and therefore the mesh is not conforming anymore.

In the last subsection, we will also motivate the choice of using higher order degrees of interpolation. We will indeed look at the best degree to obtain a given accuracy in the solution, i.e. the degree for which we have the solution to a given problem on a given mesh in the shortest amount of time and that is accurate enough.

2.3.1 No hanging nodes

Let us first present the problem we will solve in this part. We want our numerical solution to capture both low and high frequency modes so we will superpose a cosine with low frequency and a sine with a high frequency. As before, the domain is : $\Omega = [-1; 1]^2$ and Γ is the boundary. We will solve :

$$\nabla^2 u = -\frac{\pi^2}{2}\cos(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y) - 5\pi^2 \sin(5\pi x)\sin(5\pi y) \quad \text{on } \Omega \tag{2.9}$$

$$u = 0 \quad \text{on } \Gamma \tag{2.10}$$

This problem has an analytic solution that is given by :

$$u(x, y) = \cos(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y) + \frac{1}{10}\sin(5\pi x)\sin(5\pi y)$$

An example of a numerical solution obtained using the preconditioned conjugate gradients with the two scale preconditioner can be seen on figure 2.10. This solution has been obtained in only 8 iterations and with an interpolation of degree $p = 2$.

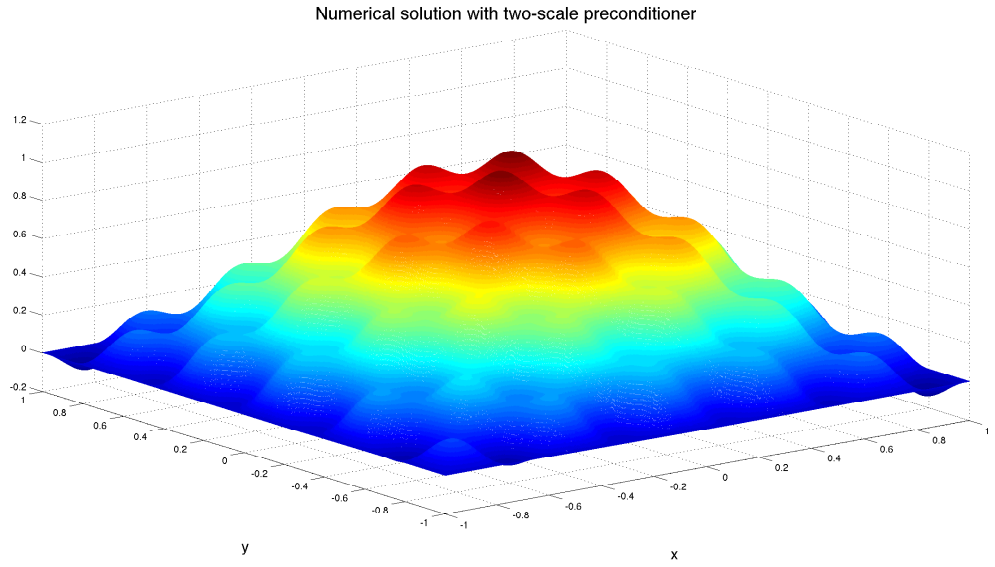


Figure 2.10: Numerical solution to problem 2.9 using an interpolation of order $p = 2$ and $1.0\,10^6$ degrees of freedom on a regular mesh with no hanging nodes. This numerical solution has been obtained by the PCG with the two scale preconditioner.

Number of quadrants	128^2	256^2	512^2	1024^2
$p = 2$	$6.6\,10^4$	$2.6\,10^5$	$1.1\,10^6$	$4.2\,10^6$
$p = 4$	$2.6\,10^5$	$1.1\,10^6$	$4.2\,10^6$	$1.7\,10^7$
$p = 6$	$5.9\,10^5$	$2.4\,10^6$	$9.4\,10^6$	$3.8\,10^7$
$p = 8$	$1.1\,10^6$	$4.2\,10^6$	$1.7\,10^7$	$6.7\,10^7$

Table 2.6: Number of degrees of freedom for a regular mesh with different number of quadrants and for different degrees of interpolation.

Regular meshes

Let us now look at what happens to the number of iterations when we decrease the mesh size for different degrees of interpolation. Table 2.6 shows the number of degrees of freedom for the different meshes used (indeed, for a given mesh, the higher the degree of the interpolation, the more global nodes we have). We can already see that, thanks to the coarse preconditioner, we are able to have a lot more degrees of freedom than in the case where we only had the fine preconditioner.

For the following tests, we also have tighten the tolerance on the norm of the residual. We now require that :

$$\frac{||r_k||_2}{||r_0||_2} < 10^{-5}$$

Figure 2.11 shows the number of iterations needed to reach that tolerance of the norm of the residual when we use our two scale preconditioner for different degrees of interpolation. The first remark we can make is that the number of iterations does not increase with the number of quandrants (as it was the case when we only had the fine preconditioner). So we can conclude that the coarse preconditioner does the job it was designed to do. We can even note that the number of iterations slightly decreases. For example, for $p = 6$, we need 12 iterations for 128^2 quadrants but we only do 11 iterations for 1024^2. An explanation for this phenomenon might be that when we increase the number of quadrants, we actually better separate the actions of the fine and coarse preconditioners and therefore the sum of the two is a better approximation of

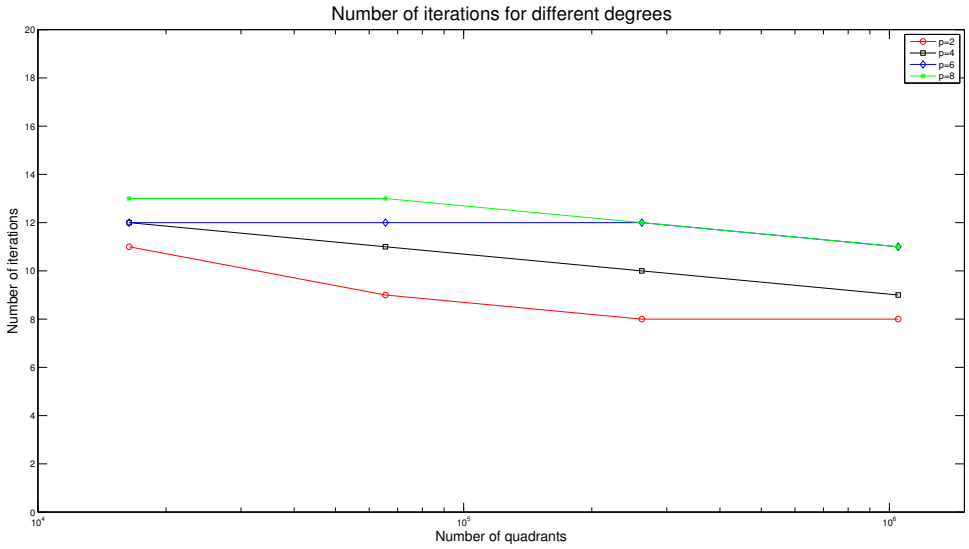


Figure 2.11: Number of iterations of PCG with the two scale preconditioner for degree p of interpolation needed to reach the given tolerance of the norm of the residual as a function of the number of quadrants in a regular mesh.

A^{-1}. We can mention that for $p = 8$, we only need 11 iterations to solve a system that has more than 67 millions unknowns.

A second remark we can make, as already observed when we only had the fine preconditioner, is that the number of iterations grows with the degree of the interpolation. For example, with 1024^2 quadrants, we need 8 iterations when $p = 2$ but we need to do 11 iterations when $p = 8$. As before, this can be explained by the fact that as the degree of the interpolation increases, the size of the overlap decreases. A fixed sized overlap would fix this issue.

Meshes with distorted elements

We will now look at what happens when the mesh is not regular but we have quadrants that are more and more distorted. We will use the same meshes already used in the previous section and obtained with the progression tool of GMSH (see figure 2.6 for example of such meshes).

For an interpolation of degree $p = 2$, we looked at the number of iterations needed to reach the given tolerance as we increase the number of quadrants and for elements more and more distorted. Figure 2.12 shows the results. We can see that the number of iterations stays roughly the same when we increase the number of quadrants so once again the coarse part of the preconditioner does the job it is designed to do.

We can also node that when the more we distort the quadrants the more iterations we need to reach the given tolerance. This is to be expected since we developed the fine preconditioner to work optimally on quadrants aligned with the axis. This means that the more a quadrant is distorted the less accurate is the fine preconditioner part and therefore the more iterations we need. However, even with the most distorted mesh, the number of iterations stays acceptable and as explained in the previous section, the gain of not having to solve exactly on every quadrant is huge. We also have to say that the meshes presented here have an intrinsic structure since they are generated using the progression tool in GMSH. Therefore, the errors we make with our approximation are always in the same direction and the number of iterations needed increases. We might not have such an increase with a random mesh containing quadrants with the same quality measure.

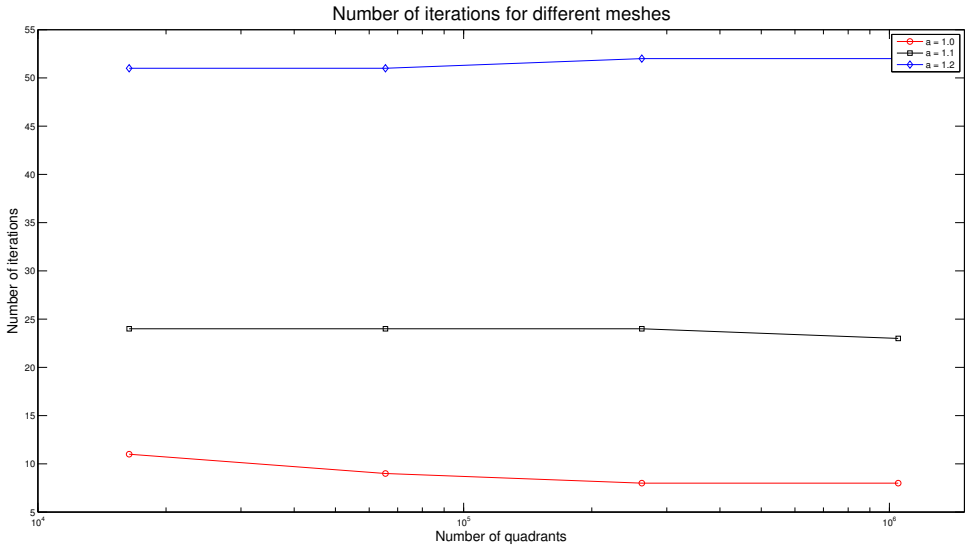


Figure 2.12: Number of iterations of PCG with the two scale preconditioner for an interpolation degree $p = 2$ needed to reach the given tolerance on the norm of the residual as a function of the number of quadrants for meshes with quadrants more and more distorted.

2.3.2 Influence of hanging nodes

Let us now move on to the cases when we have hanging nodes. As in the part with only the fine preconditioner, we will look at a problem where the solution has a jump so that we create hanging nodes when we use a recursive refine function in p4est. In addition to the hyperbolic tangent already presented in the previous subsection, we will include a high frequency sine wave to see how the grid adapts to capture it and how the two scale preconditioner performs in its presence.

The problem we will solve is :

$$\begin{aligned} \nabla^2 u = &-2\tanh(12x)\tanh(12y)\left[12^2(1-\tanh(12x)^2)+12^2(1-\tanh(12y)^2)\right] \\ &-10\pi^2\sin(10\pi x)\sin(10\pi y) && \text{on } \Omega \quad (2.11) \\ u = &\tanh(12x)\tanh(12y)+\frac{1}{20}\sin(10\pi x)\sin(10\pi y) && \text{on } \Gamma \quad (2.12) \end{aligned}$$

Where $\Omega = [-1;1]^2$ and Γ is the boundary. This problem has an analytic solution that is given by :

$$u(x,y) = \tanh(12x)\tanh(12y)+\frac{1}{20}\sin(10\pi x)\sin(10\pi y)$$

Figure 2.13 shows an example of the numerical solution computed on a non conforming mesh with PCG and the two scale preconditioner. We can see on the plot both the steep jump due to the hyperbolic tangent and the small oscillations due to the sine wave.

Increasing the relative number of hanging nodes

Let us now look at the influence of increasing the number of hanging nodes in the mesh. To achieve this, we will use the same rule in the recursive refinement of the quadrants as in the previous section, given by relation 2.7. We will also tighten the parameter *tol* used in order to have more quadrants than in the case when we only had the fine preconditioner.

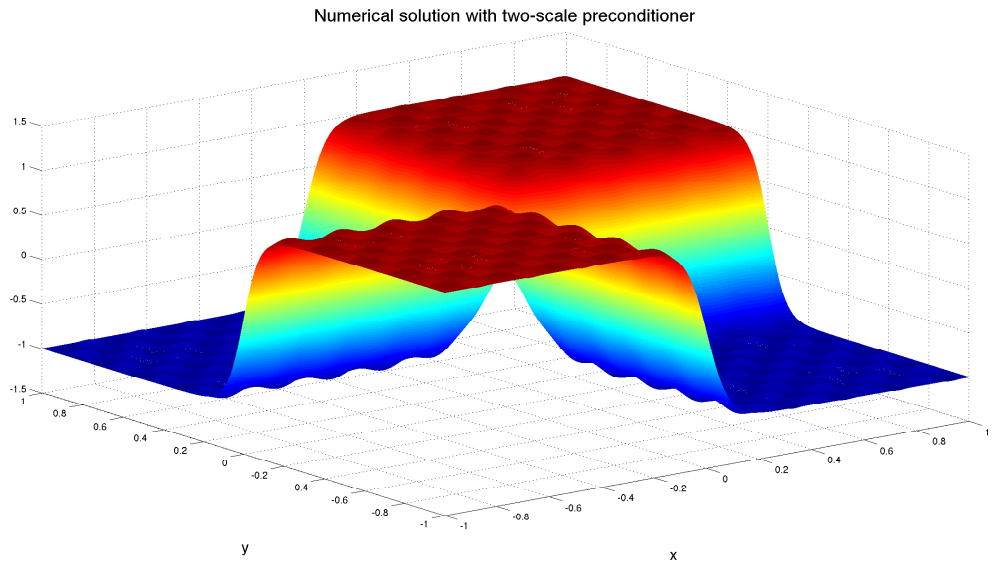


Figure 2.13: Numerical solution to problem 2.11 using an interpolation of order $p = 2$ on a non conforming mesh and obtained with PCG with the two scale preconditioner.

Parameter *tol*	0.01	0.008	0.006	0.004	0.002	0.001	0.0005
Number of hanging nodes	6080	8064	11008	14272	23200	33696	70112
Number of global nodes	19921	25545	32265	49265	120201	208137	445425
hang	30.52%	31.57%	34.12%	28.97%	19.30%	16.19%	15.74%

Table 2.7: Statistics about the different meshes used for the tests regarding the influence of hanging nodes on the number of iterations as well as the ratio *hang* defined by equation 2.8. The number of global and hanging nodes are for an interpolation degree $p = 2$.

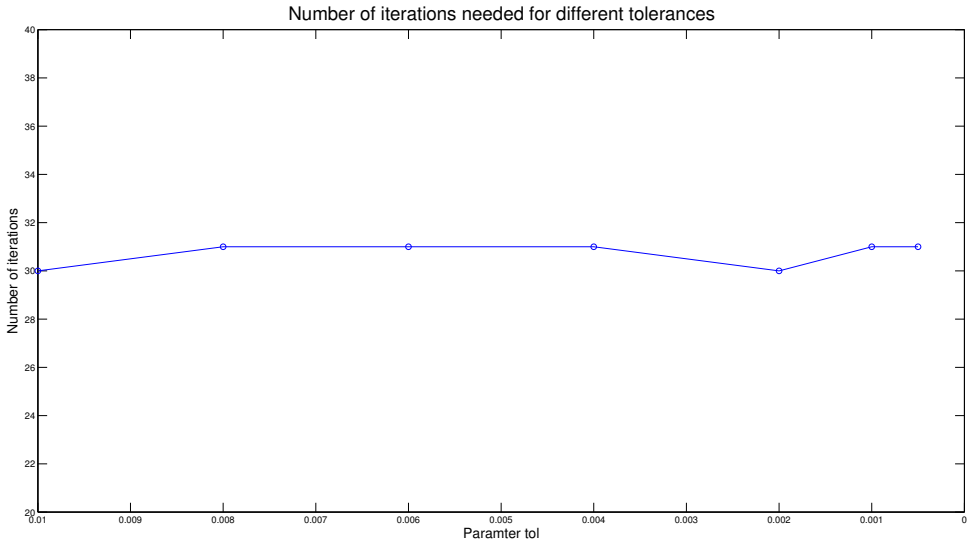


Figure 2.14: Number of iterations of PCG with the two scale preconditioner with a degree of interpolation $p = 2$ on meshes with hanging nodes defined by the parameter *tol*.

Table 2.7 shows some statistics about the different meshes used for the tests regarding the influence of hanging nodes on the number of iterations. The number of global and hanging nodes are for an interpolation of degree $p = 2$ and the ratio *hang* is given in equation 2.8.

We can see that, as expected, when we decrease the parameter *tol*, we have more hanging nodes. We can also see that, at first, tightening the parameter *tol* increase the ratio *hang* and we have more hanging nodes relatively to global nodes. Then, however, even tough the number of hanging nodes still grows, the number of global nodes grows faster and therefore the ratio *hang* decreases. This can be explained by the fact that if the parameter *tol* is very low then we start to refine all part more equally (even the ones where the function does not vary a lot) and the relative number of hanging nodes drops.

Figure 2.14 shows the number of iterations needed to reach the given tolerance with an interpolation degree $p = 2$ on the different meshes presented in table 2.7. We can see that the number of iterations stays roughly the same (it is either 30 or 31) for all meshes, however the number of hanging nodes and the value of the ratio *hang*. So we can conclude that the coarse preconditioner once again guaranties the h-independent convergence and that the number of hanging nodes does not have a great influence on the number of iterations.

We also have to note that, for the same problem, if we use a conforming mesh, the number of iterations needed drops to 16. So even if the number of hanging nodes (absolute or relative) does not greatly influence the number of iterations, just adding one non conforming quadrant does a lot to decrease the performance of the preconditioner. We can point the fact that, as mentioned earlier, when we have hanging quadrants, we do not handle every possibilities to compute the residuals in the overlaps. We can thus see how that affects the number of iterations.

Increasing the degree of the interpolation

Let us now move on to look at what happens when we increase the degree of the interpolation. We will use the mesh defined by $tol = 0.001$ and look at the number of iterations needed to reach the given tolerance for $p = 2, 4, 6, 8$. Figure 2.15 shows the results.

We can see that as we increase the degree of the interpolation, the number of iterations increases. It goes from 31 for $p = 2$ to 85 iterations when $p = 8$. We can see that it is quite a lot. But we also have to note that going from degree 2 to degree 8, we have also gone from 2.1 10^5

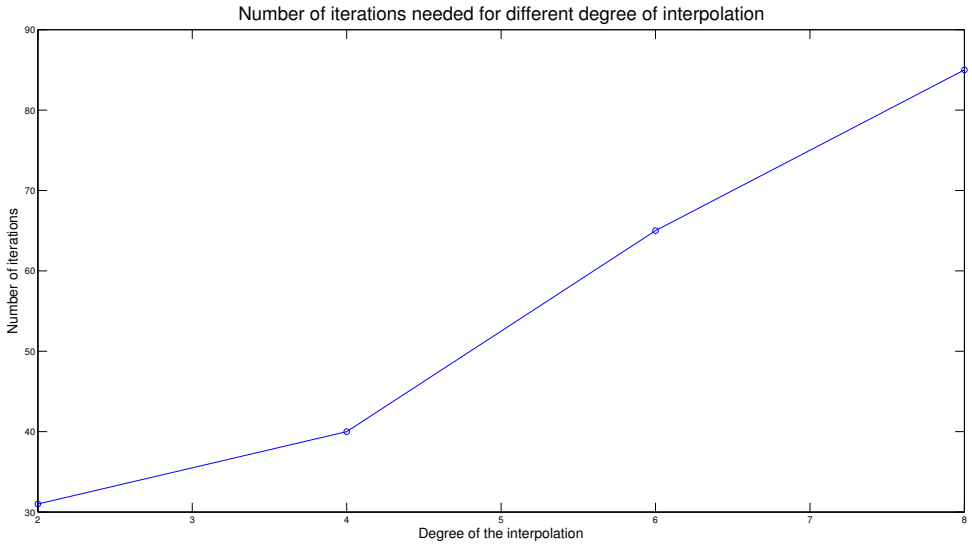


Figure 2.15: Number of iterations of PCG with the two scale preconditioner as a function of the degree of the interpolation used for a mesh obtained with $tol = 0.001$ in the recursive refine function.

degrees of freedom to $3.4\,10^6$.

As in the case when we only had the fine preconditioner, several factors can be put forward to explain the increase in the number of iterations. First, when we increase the degree, the size of the overlap decreases and therefore the fine preconditioner is less effective. As mentioned in [2], a fixed sized overlap would fix the issue.

Second, we can say that the number of hanging nodes in the overlaps increases with the degree of the interpolation. Since we do not treat all possibilities when handling hanging quadrants, the error made is more important when we have a lot of nodes in the overlap.

2.3.3 Most efficient degree to obtain a given accuracy

In this subsection, we will look at the best degree of interpolation p to obtain a given accuracy. We will use the problem defined by 2.9 and the distorted mesh using the progression tool of GMSH with $a = 1.2$ that we will uniformly refine. We will also tighten the tolerance on the norm of the residual since we want the error to come from the interpolation and not from the fact that we only solve the linear system approximately. We will impose that :

$$\frac{||r_k||_2}{||r_0||_2} < 10^{-12}$$

Afterwards, we will compute the error committed by the discretization in the L^2-norm. Let us define e^p, the error made by using an interpolation of degree p, u^p the solution of the discretized problem using a degree p, U_i^p the numerical solution at the global node i of the linear system that arises and m_i the mass matrix associated with the integration. Then we have :

$$e^p = \left(\int_\Omega (u - u^p)^2 dxdy \right)^{\frac{1}{2}} \tag{2.13}$$

$$\approx \left(\sum_i (u(x_i, y_i) - U_i^p)^2 m_i \right)^{\frac{1}{2}} \tag{2.14}$$

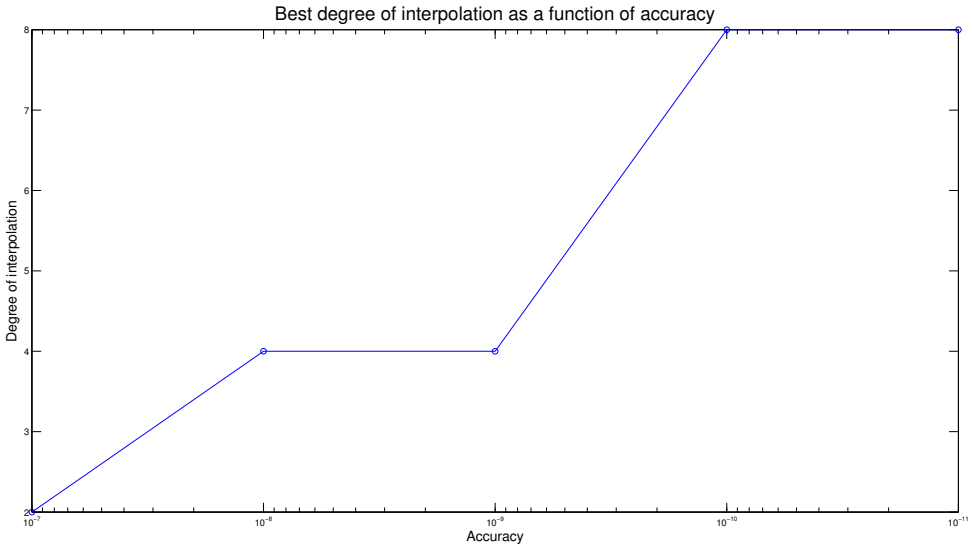


Figure 2.16: Best degree of interpolation to solve problem 2.9 on a distorted mesh as a function of the accuracy we want on the L^2-norm.

We can note that if we refine uniformly the mesh, then e^p will decrease but it will also cost us more. And for a given mesh, we have that e^p decreases as p increases (but it also costs us to increase the degree). So the question that arises is when is it more interesting to refine the mesh uniformly and when is it better to increase the degree of the interpolation to get the accuracy we want.

We will select degree p to be the best for an accuracy of 10^{-a} if and only if there is a mesh such that $e^p < 10^{-a}$ on this mesh and the time needed to obtain the numerical solution is smaller than for any degree on any other mesh and the same accuracy.

Figure 2.16 shows the results. We can see that when the wanted accuracy is rather low (for 10^{-7}) then it is quicker to use an order of interpolation $p = 2$ because the number of iterations needed to solve the linear system is smaller than for a higher order interpolation. Moreover, the number of degrees of freedom to obtain such an accuracy is not so high as to make one iteration very long.

On the other hand, when we want a better accuracy (10^{-10}, 10^{-11}) then it is cheaper to use higher order interpolation such as $p = 8$. This can be explained by the fact that, even though the number of iterations of PCG is higher (as mentioned earlier in this section), the number of degrees of freedom to obtain the wanted accuracy is much lower for $p = 8$ than for the other degrees. Therefore, one iteration takes much less time and $p = 8$ becomes more efficient.

In between the two (10^{-8}, 10^{-9}), we can see that the best degree is $p = 4$ where we have a trade-off between the number of iterations needed to solve the system and the number of degrees of freedom.

The quantitative results presented here depend of course of the implementation, the problem to solve and the mesh used. However, we can note that the qualitative result stays true : the more accurate we want to be, the more interesting it becomes to use higher degree in the interpolation because then the number of DOF does not grow as large as it would with lower order polynomials.

2.4 Conclusion

Let us now summarize the results given in this chapter.

We first looked at the properties of our geometric multigrid solver. The key property of h-independent convergence has been observed for a large variety of parameters ν_1 and ν_2 and

various meshes that were regular or distorted. The presence of hanging nodes has no influence whatsoever on the convergence of the geometric multigrid method.

We then moved on to the PCG with only the fine scale preconditioner. Several observations were made. We saw that, even on a regular mesh where the preconditioner is optimal, increasing the number of quadrants increased the number of iterations needed. That was predicted since increasing the number of quadrants decreases the mesh size. We also noticed that we needed more iterations when the degree of interpolation p grew. That was explained by the fact that the size of the overlap diminishes when p increases. We then move on to meshes with distorted quadrants and saw that we needed more iterations than in the regular case. It was to be expected since the fine preconditioner was designed to work best on quadrants aligned with the axes. Afterwards, we introduced non conforming meshes. The presence of hanging nodes also increases the number of iterations. That was explained by the fact that we do not restrict the residual exactly in the overlaps. When we increase the degree of the interpolation on a non conforming, the number of iterations increases as in the conforming case.

The coarse grid correction was then added in the preconditioner. We first tested the PCG with the two scale preconditioner on conforming meshes and saw that we obtained the h-independence thanks to the coarse preconditioner : the number of iterations stayed constant whatever the number of quadrants and the mesh size. That was true for both meshes with regular quadrants and meshes with distorted quadrants. We can however note that the number of iterations needed was greater in the case of distorted quadrants. Again, this was to be expected since the fine preconditioner works better on quadrants aligned with the axes. We then experimented on non conforming meshes. We saw that changing the absolute or the relative number of hanging nodes in the mesh had no real impact of the number of iterations. That being said, we saw that we needed more iterations when we had hanging nodes (even a few). We also looked at what happened when we increased the degree of the interpolation. As before, it increases with the degree p. The same remarks as before also apply here.

Finally, we picked a problem and a mesh and looked at the best degree of interpolation to solve this problem, i.e. the degree which allowed us to get the solution within a given tolerance in the L^2-norm the quickest. We saw that at first, for a low tolerance, a low degree is best since then the fine preconditioner is cheap to compute and we can have a lot of degrees of freedom. The number if iterations of PCG is also lower for lower degrees. Then, as we tighten the tolerance, it becomes more and more attractive to use higher degrees since they allow us to keep the number of degrees of freedom down and still get a good enough approximation to be within the tolerance. The increases in the number of iteration for higher degrees is compensated by the fact that we have a lot less degrees of freedom. This means that for a given tolerance, we have a trade-off between the number of iterations of PCG and the number of degrees of freedom.

Bibliography

[1] LAPACK - Linear Algebra PACKage. `http://www.netlib.org/lapack/`. Accessed: 2017-04.

[2] Luca F Pavarino. Additive Schwarz methods for the p-version finite element method. *Numerische Mathematik*, 66(1):493–515, Dec 1993.

[3] GMSH. `http://gmsh.info/`. Accessed: 2017-04.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**