Mass Lumped triangular elements for the Wave Equation

Daniele Ceccarelli and Tommaso Missoni
Supervisor: Dr. I. Mazzieri

February 2020

Abstract

The main goal of this project is to implement a Mass Lumping technique for finite element approximations up to third order in space to solve the 2D scalar wave equation with Dirichlet boundary conditions.
We have chosen to develop this method to avoid the problem of inverting a full mass matrix at each time step, since with this technique the mass matrix is diagonal. Moreover, we have adopted a triangular mesh for the flexibility in modeling complex geometries.
In Section 1 we present the model problem and the theory of Mass Lumping technique, in Section 2 we describe the coding implementation, and in Section 3 we test our algorithm on some interesting problems.

Contents

1 Introduction

Solving the wave equation in the time domain by numerical methods is a delicate but fundamental problem for modeling numerous physical phenomena such as acoustic, elastic, or electromagnetic waves. For such phenomena, the wave equation serves as a model problem.

The numerical approximation of this equation is essential, in particular, for two cases for which analytical methods do not apply:

- heterogeneous media;
- domains of arbitrary shape.

The most popular methods to solve this problem are the Finite Element Method (FEM) and the Spectral Element Method (SEM).
The FEM can be used on very complex geometries, thanks to the employment of a triangular mesh, that can approximate well most of the realistic domains; moreover, it is quite simple to implement. On the other hand, it is computationally expensive because we need to invert a full mass matrix at every time step.
On the contrary, the SEM is very fast in terms of computational time, thanks to the use of a diagonal mass matrix, but a quadrilateral mesh is needed, and this can be a problem even on a not so complex domain.
In an effort to keep the good properties of both methods, a Mass Lumping technique on a triangular mesh was created.
The aim of this method is to achieve a good accuracy on a triangular mesh, implementing a finite element method with a diagonal mass matrix. Usually, this is done with nodal basis functions and an inexact quadrature rule for the mass matrix, M, in which the quadrature points coincide with the basis functions nodes.
Generally, mass lumping is useful for solving any type of evolution problem that requires the inversion of a mass matrix. Anyway, this project focuses on the 2D scalar wave equation with fully Dirichlet boundary conditions.

1.1 Model problem

1.1.1 Strong and weak formulation

We consider the following problem:

find $u : \Omega \times (0,T) \to \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2}(\mathbf{x},t) - \nabla \cdot (k(\mathbf{x})\,\nabla u(\mathbf{x},t)) = f(\mathbf{x},t) & (\mathbf{x},t) \in \Omega \times (0,T) \\ u(\mathbf{x},0) = u_0(\mathbf{x}) & \mathbf{x} \in \Omega \\ \dfrac{\partial u}{\partial t}(\mathbf{x},0) = u_1(\mathbf{x}) & \mathbf{x} \in \Omega \\ u(\mathbf{x},t) = g(\mathbf{x},t) & (\mathbf{x},t) \in \partial\Omega \times (0,T) \end{cases} \tag{1}$$

where $\Omega \subseteq \mathbb{R}^2$ is a Lipschitz domain, $u : \Omega \times (0,T) \to \mathbb{R}$ is the unknown scalar field, $f : \Omega \times (0,T) \to \mathbb{R}$ is the source term and $k : \Omega \to \mathbb{R}^+$ is a positive scalar field.
We assume that $k_0 \leq k \leq k_1$ for some $k_0, k_1 \in \mathbb{R}^+$. This hypothesis follows from physical considerations, since the speed of propagation of the wave is $\sqrt{k}$. Moreover, from the theoretical point of view, we assume that: $u_0 \in H^1(\Omega)$, $u_1 \in L^2(\Omega)$, $g(t) \in H^{1/2}(\Omega)\, a.e.\, t \in [0,T]$, $f \in L^2(0,T;L^2(\Omega))$.

By integration by parts we obtain the weak formulation of this problem:

find $u \in L^2\left(0,T;H^1(\Omega),L^2(\Omega)\right)$ s.t. $u_t \in L^2(0,T;L^2(\Omega))$, $u_{tt} \in L^2(0,T;H^1(\Omega)^*)$ and

$$\int_\Omega \frac{\partial^2 u}{\partial t^2}(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} + \int_\Omega (k(\mathbf{x})\, \nabla u(\mathbf{x},t)) \cdot \nabla v(\mathbf{x})\, d\mathbf{x} = \int_\Omega f(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} \quad (2)$$

a.e. $t \in (0,T)$, $\forall v \in H_0^1(\Omega)$.
Note that the boundary integral which follows from the integration by parts, $-\int_{\partial\Omega} (k(\mathbf{x})\, \nabla u(\mathbf{x},t)\, v(\mathbf{x})) \cdot \mathbf{n}\, d\sigma$, disappears because of the hypotheses made for v.
We choose the test function v in H_0^1 even if we consider non-homogeneous Dirichlet boundary conditions. In order to recover the same space for u and v we need to define a lifting operator $G : H^1(\Omega) \to H^1(\Omega)$ such that

$$(Gu)(\mathbf{x},t)|_{\partial\Omega} = g(\mathbf{x},t)\ \forall t \in [0,T].$$

Setting now $\tilde{u} = u - Gu$, we obtain that $\tilde{u} \in H_0^1$ and the problems for $\tilde{u}$ are

find $\tilde{u} : \Omega \times (0,T) \to \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial^2 \tilde{u}}{\partial t^2}(\mathbf{x},t) - \nabla \cdot (k(\mathbf{x}) \nabla \tilde{u}(\mathbf{x},t)) = f(\mathbf{x},t) - \dfrac{\partial^2 (Gu)}{\partial t^2}(\mathbf{x},t) + \nabla \cdot (k(\mathbf{x}) \nabla (Gu)(\mathbf{x},t)) & (\mathbf{x},t) \in \Omega \times (0,T) \\ \tilde{u}(\mathbf{x},0) = u_0(\mathbf{x}) - (Gu)(\mathbf{x},0) & \mathbf{x} \in \Omega \\ \dfrac{\partial \tilde{u}}{\partial t}(\mathbf{x},0) = u_1(\mathbf{x}) - \dfrac{\partial (Gu)}{\partial t}(\mathbf{x},0) & \mathbf{x} \in \Omega \\ \tilde{u}(\mathbf{x},t) = g(\mathbf{x},t) & (\mathbf{x},t) \in \partial\Omega \times (0,T) \end{cases} \quad (3)$$

and

$$\int_\Omega \frac{\partial^2 \tilde{u}}{\partial t^2}(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} + \int_\Omega (k(\mathbf{x})\, \nabla \tilde{u}(\mathbf{x},t)) \cdot \nabla v(\mathbf{x})\, d\mathbf{x} =$$

$$\int_\Omega f(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} - \int_\Omega \frac{\partial^2 (Gu)}{\partial t^2}(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} - \int_\Omega (k(\mathbf{x})\, \nabla (Gu)(\mathbf{x},t)) \cdot \nabla v(\mathbf{x})\, d\mathbf{x}$$

but for the sake of simplicity we use just u in the following pages.
Notice that in the previous weak formulation we have a second-order time deriva-

tive inside the spatial integral, so we will need two steps to discretize this problem: we first obtain the semi-discretization in space and then we discretize in time to obtain the fully-discretized form.

We define now the bilinear form

$$a(u,v) = \int_\Omega k(\mathbf{x})\, \nabla u(\mathbf{x},t) \cdot \nabla v(\mathbf{x})\, d\mathbf{x}$$

and the linear operator

$$F(v) = \int_\Omega f(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x}$$

so we can rewrite the problem as

$$\int_\Omega \frac{\partial^2 u}{\partial t^2}(\mathbf{x},t)\, v(\mathbf{x})\, d\mathbf{x} + a(u,v) = F(v)\,. \tag{4}$$

1.1.2 Well-posedness

Theorem 1.1 *With the above assumptions on f, u_0, u_1, and assuming $g = 0$, the weak problem has a unique solution and*

$$||u||^2_{L^\infty(0,T;H^1(\Omega))} + ||u_t||^2_{L^\infty(0,T;L^2(\Omega))} + ||u_{tt}||^2_{L^2(0,T;H^1(\Omega)^*)} \leq \\ \leq C\{||\nabla u_0||^2_0 + ||u_1||^2_0 + ||f||^2_{L^2(0,T;L^2(\Omega))}\} \tag{5}$$

with $C = C(k,T,\Omega)$.

For the proof, see e.g. [8], [6].
Note that the result for $g \neq 0$ follows immediately using the lifting operator.

1.1.3 Discrete formulation

Spatial semi-discretization

Let $V_h \subseteq H^1(\Omega)$, such that $dim\, V_h = N_h < \infty$.
Then the semi-discretization of the weak problem (2) gets the following form:

for any $t \in (0,T)$, find $u_h = u_h(t) \in V_h$ s.t.

$$\begin{cases} \displaystyle\int_\Omega \frac{\partial^2 u_h}{\partial t^2}(t)\, v_h\, d\Omega + a(u_h(t), v_h) = F(v_h) \qquad \forall v_h \in V_h,\ a.e.\, t \in (0,T) \\ u_h(\mathbf{x},0) = \Pi_h u_0(\mathbf{x}) \qquad \mathbf{x} \in \Omega \\ \displaystyle\frac{\partial u_h}{\partial t}(\mathbf{x},0) = \Pi_h u_1(\mathbf{x}) \qquad \mathbf{x} \in \Omega \end{cases} \tag{6}$$

where $\Pi_h u_0$ and $\Pi_h u_1$ are suitable approximations of the initial conditions (e.g., Π_h may represent the projection of $L^2(\Omega)$ onto V_h).

Algebraic formulation

Since V_h is finite dimensional, we can introduce a basis $\{\varphi_i(\mathbf{x})\}_{i=1}^{N_h}$, and so we can find N_h coefficients $u_i \in L^2([0,T])$ s.t.

$$u_h(\mathbf{x},t) = \sum_{i=1}^{N_h} u_i(t)\,\varphi_i(\mathbf{x})\ .$$

Moreover, since every element of V_h is a linear combination of the basis functions, and the problem is linear in v_h, now it is sufficient that the semi-discrete problem (6) holds only for every basis function φ_i, instead of for every $v_h \in V_h$.

We introduce now the bilinear form

$$m(u,v) = \int_\Omega u(\mathbf{x},t)\,v(\mathbf{x})\,d\mathbf{x}\,. \tag{7}$$

Using these definitions in the semi-discrete formulation (6) we get

find $\{u_i\}_{i=1}^{N_h}$ s.t.

$$\sum_{i=1}^{N_h} \frac{\partial^2 u_i}{\partial t^2} m(\varphi_i,\varphi_j) + \sum_{i=1}^{N_h} u_i a(\varphi_i,\varphi_j) = F(\varphi_j)\ \forall j = 1,...,N_h\,. \tag{8}$$

Introducing now the matrices

- $M = [m_{ij}] = m(\varphi_j,\varphi_i)$ **mass matrix**
- $A = [a_{ij}] = a(\varphi_j,\varphi_i)$ **stiffness matrix**

and the vectors

- $F = [F_j] = F(\varphi_j)$ **forcing vector**
- $U = [u_1 ... u_{N_h}]^T$ **unknowns vector**

the problem (6) can be written as

find $U \in \mathbb{R}^{N_h}$ s.t.

$$\begin{cases} M\dfrac{\partial^2 U}{\partial t^2} + AU = F \quad \forall t \in (0,T) \\ U(0) = U_0 \\ \dfrac{\partial U}{\partial t}(0) = \dot{U}_0 \end{cases} \tag{9}$$

with

$$U_{0,i} = \int_\Omega \Pi_h u_0\,\varphi_i\,d\Omega \quad \dot{U}_{0,i} = \int_\Omega \Pi_h u_1\,\varphi_i\,d\Omega\,,$$

which is a linear differential algebraic system in the unknown U.

Temporal discretization

Now, the temporal discretization consists in discretizing the time domain and substituting the second order time derivative of U and the initial condition for its first order derivative with suitable approximations.

Let us subdivide the time interval $(0, T]$ into N_t subintervals of ampitude Δt and consider the time instants $t^n = n\,\Delta t \;\; \forall n = 1, ..., N_t$.
In this work, to integrate in time problem (9), we consider the leap-frog scheme; hence, the fully discrete problem becomes

find $U^{n+1} \in \mathbb{R}^{N_h}$ s.t.

$$\begin{aligned} MU^1 &= (M - \frac{\Delta t^2}{2}A)U_0 + \Delta t M\dot{U}_0 + \frac{\Delta t^2}{2}F^0 \\ MU^{n+1} &= (2M - \Delta t^2 A)U^n - MU^{n-1} + \Delta t^2 F^n \;\; \forall n = 1, ..., N_t \end{aligned}$$

with $U^n \approx U(t^n)$ and $F^n \approx F(t^n)$.
Apart from the first step, the solution of this system is formally given by

$$U^{n+1} = 2U^n - U^{n-1} + \Delta t^2 M^{-1}(F^n - AU^n)\,.$$

We observe that, at the end, the key point for the computation of U^{n+1} is the inversion of the mass matrix, M.

1.1.4 Error analysis

From [1] and [3], we have the following result for the error:

Theorem 1.2 *Under suitable conditions on the solution u (at least $u \in H^r$) and on Ω, f, g, u_0 and u_1, we have that:*

$$||u - u_h||_{H^m} \leq C(h^{r+1-m} + dt), \;\; m = 0, 1$$

where u_h is the discrete solution.

1.2 Mass Lumping

1.2.1 Theoretical background

Mass Lumping technique is used when we want to compute the integral (7): in particular, the aim of this approach is to have a diagonal mass matrix, in order to make its inversion straightforward.
If we introduce the basis functions $\{\varphi_i\}$, the computation of M is made through

$$M_{ij} = \int_\Omega \varphi_i(\mathbf{x})\, \varphi_j(\mathbf{x})\, d\mathbf{x} = (\varphi_i, \varphi_j)_{L^2}.$$

and so it is diagonal if and only if

$$(\varphi_i, \varphi_j)_{L^2} = 0 \quad \forall i \neq j.$$

Unfortunately, this is not true in general: not all the possible sets of basis functions are L^2-orthogonal.
From now on, we decide to proceed with a Lagrangian basis, with the interpolation nodes $\{\mathbf{a}_i\}$.
If now, instead of computing the integrals defining M_{ij} analytically, we evaluate them approximately, using a quadrature formula in each triangle $K \in \mathcal{T}_h$, we in fact replace the continuous L^2 inner product $(\varphi_i, \varphi_j)_{L^2}$ by the discrete inner product $(\varphi_i, \varphi_j)_h$, defined by

$$M^h_{ij} = (\varphi_i, \varphi_j)_h = \sum_{l=1}^{N} w_l\, \varphi_i(\hat{\mathbf{a}}_l)\, \varphi_j(\hat{\mathbf{a}}_l),$$

where $w_l \in \mathbb{R}$ are the quadrature weights and $\hat{\mathbf{a}}_l \in \Omega$ are the quadrature nodes of the formula, and so M^h_{ij} is the numerical approximation of M_{ij}.
We can have a lumped mass matrix if we choose a quadrature formula such that

$$(\varphi_i, \varphi_j)_h = 0 \, \forall i \neq j \,,$$

so that our basis function are orthogonal with respect to the discrete inner product we have defined. To choose a proper quadrature formula we make use of the following fundamental lemma (see [1]):

Lemma 1.3 *If the nodes of the finite element space V_h and the quadrature points coincide, i.e., if*

$$\{\mathbf{a}_i\} = \{\hat{\mathbf{a}}_l\} \tag{10}$$

then one has mass lumping.

The previous lemma is based on the fact that $\varphi_i(\mathbf{a}_j) = \delta_{ij}$, (due to the fact that we are using a Lagrangian basis) and that if the nodes of V_h and the quadrature nodes coincide, in the sum of the discrete inner product the terms are equal to zero if $i \neq j$ (because φ_i and φ_j are different from zero on different nodes):

$$(\varphi_i, \varphi_j)_h = \sum_{l=1}^{N} w_l \, \varphi_i(\hat{\mathbf{a}}_l) \, \varphi_j(\hat{\mathbf{a}}_l) = \sum_{l=1}^{N} w_l \, \delta_{il} \, \delta_{jl} = \delta_{ij} \, w_i = \delta_{ij} \, w_j.$$

In addition to the diagonal property, we want our new quadrature formula to satisfy this Accuracy condition ($\mathcal{A}$):
($\mathcal{A}$): In each triangle K, the quadrature formula must be exact for P_{2s-2}, where P_s is the space of polynomials of degree up to s.

1.2.2 Pros and cons

The main goal of the Mass Lumping approach is to speed up the computation of the mass matrix, also reducing the computational cost: a linear system in the mass matrix has to be solved at each time step, and a diagonal mass matrix would signify a huge gain in terms of computational time.
However, we do not expect the mass lumped quadrature rule to be the best one available, so we may have a less accurate result: the point is to find the best trade-off between accuracy and computational speed.
Moreover, the choice of a triangular mesh is justified by the fact that we want to have the possibility to model domains of quite complex shape, considering the practical applications that this problem has: this is not achieved in the best way adopting, e.g., a quadrilateral mesh.
We notice that this is just one of the methods used to obtain a diagonal mass matrix. In particular, we can find three different ways to implement Mass Lumping:

1. **row sum method**: if we compute the mass matrix with a traditional continuous finite element method, we can compute a lumped matrix just by summing all the elements of the $i-th$ row and put the result at the $i-th$ position:
$$M_{ii}^{lumped} = \sum_j M_{ij} \, ;$$

2. **diagonal scaling**:
$$M_{ii}^{lumped} = c M_{ii} \, ,$$
with c adjusted to satisfy
$$\sum_j M_{jj}^{lumped} = \sum_i \sum_k M_{ik} \, ,$$
which is a sort of "conservation of mass".

3. **nodal points method**: this is the method presented in this work, and consists in using a quadrature rule in the nodal points; to avoid numerical problems such as negative quadrature weights we need to enrich the FE space.

Our method, compared with the other two, is in some way more complicated (since we need to define a new finite element space). However, it is more accurate, because the other two approximate the mass matrix in a simpler way.

2 Implementation

For our work, we chose as starting point a code, available in [4], which was able to solve the Poisson problem. This code can solve the Poisson equation with both Dirichlet and Neumann boundary conditions, and also mixed problems, using a FE method on a triangular mesh with polynomial basis functions of arbitrary degree. Anyway, the quadrature rule that this code uses to assembly the matrices of the algebraic system (and in particular the mass matrix) is not the suitable one to implement the Mass Lumping method. Moreover, the available domains in which the code can solve the problem are only the square and the circle. So, to adapt this code to our goals, we had to:

- implement the time step evolution;
- incorporate our almost arbitrary mesh in the code;
- create our new Finite Element space for Mass Lumping;
- create from zero the assembly of the mass matrix with our suitable quadrature rule.

Note that the last point involves also the right choice for the nodes of the FE method: as already shown in Section 1.2.1, in order to have Mass Lumping the degrees of freedom of the discrete space V_h and the quadrature nodes must coincide.

2.1 Time discretization

As we have already seen, we use a leap-frog scheme for the discretization of the time derivative. This is a first order method, explicit but conditionally stable (a CFL condition must be respected).

$$\Delta t < Ch \quad (CFL\ condition)$$

A possible alternative of the same order of accuracy is, e.g., the Backward Euler scheme: it is still a first order method, but it is unconditionally stable, which means that we have no condition on the time step to respect in order to have stability. The main problem is that it is an implicit method, and in particular it requires at every time step the solution of a linear system which involves the inversion of the stiffness matrix A. We simply wanted to avoid that.

We observe that also the linear approximation for the first step is of first order.

We refer to the further developments (Section 4.2) for the implementation of a higher-order scheme, noticing that a single step of a first order one-step method (such as, e.g., the Euler scheme), which is needed for multi-step schemes, will not affect the global higher-order time convergence of the scheme.

2.2 Mesh

As already said before, we chose to adopt a triangular mesh, $\mathcal{T}_h$, in order to exploit its flexibility to model quite complex domains.
We initially generate the domain and initialize the mesh through the MATLAB PDE Toolbox, which allows to design a domain with almost arbitrary shape and to obtain some informations about the geometry of the problem. In particular, one can export (see MATLAB documentation [7] for more):

- `p` (points), a $2 \times N_p$ matrix of nodes, where N_p is the number of nodes in the mesh; each column `p(:,k)` consists of the x-coordinate of point k in `p(1,k)` and the y-coordinate of point k in `p(2,k)`;
- `e` (edges), a $7 \times N_e$ matrix of edges, where N_e is the number of edges in the mesh; the mesh edges in `e` and the edges of the geometry have a one-to-one correspondence, so each column in the `e` matrix represents one edge; each column contains various information about the correspondent element, for example the indices of the two extrema of the edge, or the indices of the two triangles that share this segment as edge (the index is 0 if it is in the exterior part of Ω);
- `t` (triangles), a $4 \times N_t$ matrix of triangles, where N_t is the number of triangles in the mesh; each column of `t` contains the indices of the points in `p` that form the triangle and the subdomain number.

We designed a code able to receive these matrices as input; the output is a structure whose most important fields are:

- `Nodes`, a $N_p \times 2$ matrix, containing the coordinates of the nodes;
- `NodePtrs`, `CNodePtrs`, `FNodePtrs`, matrices containing the indices of the nodes, the constrained nodes (i.e. boundary nodes) and the free nodes (i.e. internal nodes) respectively;
- `Edges`, a $N_e \times b$ matrix, where b is the number of nodes on each edge (degree+1 in the general case); the row `Edges(k,:)` contains the indices of the b nodes of the edge k;
- `Elements`, a $N_t \times 3$ matrix; the row `Elements(k,:)` contains the indices of the 3 edges of the triangle k;
- `EdgeEls`, a $N_e \times 2$ matrix; the row `EdgeEls(k,:)` contains the indices of the 2 triangles which share the edge k.

2.3 Basis functions

Our starting point is to consider as the approximation subspace V_h the space of Lagrange finite elements subordinate to the mesh $\mathcal{T}_h$. Basically, the basis functions we are considering are the piecewise polynomials up to a certain degree

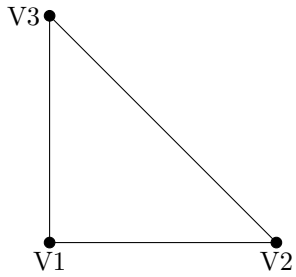


Figure 1: Ref. Triangle of P_1

p with value 1 on a certain node and 0 on all the others. So, we wish to approximate the solution of the weak problem in the space

$$V_h = \{u \in H^1(\Omega) : u|_K \in \mathbb{P}^p \, \forall K \in \mathcal{T}_h\}.$$

2.4 Quadrature rule

Using Lemma 1.3 we are sure that our mass matrix is diagonal, but we can prove that for space elements of order bigger than 1 we have some numerical problems. We lose accuracy in integration (condition $\mathcal{A}$) due to some instabilities of the quadrature formula, since the weights used become less than or equal to zero in some nodes. Let us first see the example of degree 1, where everything works without any change.

2.4.1 Degree 1

For the case of degree 1, if we want to apply the Lemma 1.2, we can choose as nodes for our quadrature formula $\{\mathbf{a}_i\} = \{\hat{\mathbf{a}}_l\}$ the three vertices of the reference triangle $\{V_1, V_2, V_3\}$, and associate to this points a weight of $\frac{1}{3}$ each. With this choice, we end up with weights that are all positive and we do not have numerical issues.

2.4.2 Problem of degree 2

For the case of P_2 elements space, we can easily find a set of nodes and weights that satisfy ($\mathcal{A}$). We know that the space of P_2 on a triangular element is made of 6 Lagrange interpolation nodes, the 3 vertices and the 3 middle-points of the edges, while a classical choice for the quadrature rule is to use just the three middle-points with a weight of $\frac{1}{3}$ each. Since now we want to use the Lemma 1.3, i.e. we want the set of the nodes of the finite element space and the quadrature points to coincide, we have to use all the six points in the quadrature rule, giving to the vertices zero weight, and to middle point a weight of 1/3 (and we can also prove that this is the only choice).

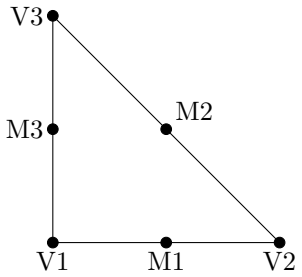


Figure 2: Ref. Triangle of P_2

$\{V_1, V_2, V_3\}$ vertexes, $\{M_1, M_2, M_3\}$ middle edge

$$I_K^{num}(f) = meas(K)\{w_v \sum_{i=1}^{3} f(V_i) + w_m \sum_{i=1}^{3} f(M_i)\}$$

where $w_v = 0$ and $w_m = \frac{1}{3}$.

As we have already said, the fact that some weights are less than or equal to zero causes some numerical problems: we need a new finite element space such that the new quadrature formula has all positive weights.

2.4.3 New finite element space for degree 2

To overcome this problem, we need to define a new space: one idea could be to increase the dimension. Let us find a space $\tilde{P}_2$, $P_2 \subseteq \tilde{P}_2 \subseteq P_3$, that satisfies these conditions:

(i) the space $\tilde{P}_2$ should be as small as possible;

(ii) we need to satisfy Lemma 1.3;

(iii) the quadrature formula should be exact in P_{2+3-2};

(iv) the weights are all strictly positive.

Let us define

$$\tilde{P}_2 = P_2 \oplus bubble$$

where $bubble = xy(1 - x - y)$ is the bubble function for the reference triangle with vertices $(0, 0)$, $(1, 0)$, $(0, 1)$. The dimension of the new discrete space is 7, so we need to add a new degree of freedom: the interior node $(\frac{1}{3}, \frac{1}{3})$ is chosen. Since we want also to satisfy Lemma 1.3, we have to choose as quadrature nodes the same Lagrangian interpolation nodes: $\{V_1, V_2, V_3\}$ vertices, $\{M_1, M_2, M_3\}$ middle-edge points and G interior node.
So, the new quadrature rule is

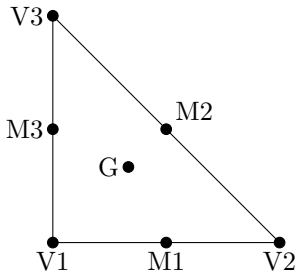


Figure 3: Ref. Triangle of $\tilde{P}_2$

$$I_K^{num}(f) = meas(K)\{w_v \sum_{i=1}^{3} f(V_i) + w_m \sum_{i=1}^{3} f(M_i) + w_g f(G)\}$$

For the choice of weights, the last two conditions are used:

Lemma 2.1 *The unique choice of weights that satisfy conditions (iii) and (iv) is to set:*

$$w_v = \frac{1}{20},\ w_m = \frac{2}{15}\ \ and\ \ w_g = \frac{9}{20}.$$

This new space $\tilde{P}_2$ has a dimension of 7, so we satisfy the first condition in an optimal way; moreover, all the other conditions are satisfied: we can consider the space

$$V_h = \{v \in H^1(\Omega)\ \ \forall K \in \tau_h,\ \ v|_K \in \tilde{P}_2\}$$

as our new approximation space for $H^1(\Omega)$.

2.4.4 General quadrature rule

We can prove that there are problems also for degree 3 (where we usually have 10 points), with some weights that become negative. The solution is equivalent to the previous one: in this case we add 2 interior points, ending up with a total of 12 nodes, as we can see in Fig. 4.

We have summed up all the quadrature rules that we used in the following table:

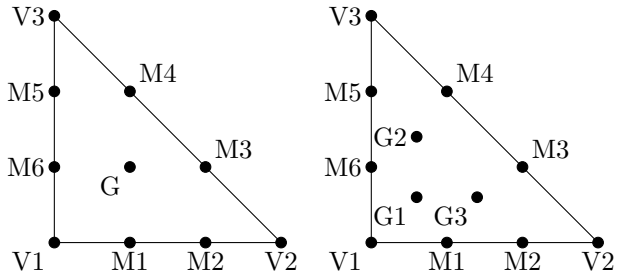


Figure 4: Ref. Triangles of P_3 and $\tilde{P}_3$

M	N	Nodes	Number	Weights	Parameter
1	1	(0,0)	3	$\frac{1}{3}$	
2	3	(0,0)	3	$\frac{1}{20}$	
		(0,0.5)	3	$\frac{2}{15}$	
		(1/3,1/3)	1	$\frac{9}{20}$	
3	4	(0,0)	3	$\frac{1}{45} - \frac{\sqrt{7}}{360}$	
		(α,0)	6	$\frac{7}{360} + \frac{\sqrt{7}}{90}$	$\alpha = \frac{1}{2} - \frac{\sqrt{(441-84(7-\sqrt{(7)})}}{42}$
		(β,β)	3	$\frac{49}{180} + \frac{7\sqrt{7}}{360}$	$\beta = \frac{1}{3}(1 - \frac{1}{\sqrt{(7)}})$

2.5 Mass Lumping shape functions

In order to build these new finite element space in our MATLAB code, we created new functions to manage the generation and evaluation of both shape functions and their gradients.
The function `shape_functions_masslumping.m` manages all this stuff, returning the evaluation of mass lumping shape functions and their gradients in some points that we named `qpts`, that can be equal to the mass lumping nodes of the triangle, but can also be different (for example in the error analysis, where we use the Dunavant Nodes).

Shape functions when d=1:

$$\varphi_i(x,y) = a_1 + a_2 x + a_3 y, \quad \forall i = 1,2,3$$

Shape functions when d=2:

$$\begin{aligned} \varphi_i(x,y) =& a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 y^2 + \\ &+ a_6 xy + a_7 \mathcal{B}_{1,2}(x,y) \quad \forall i = 1,\dots,7 \\ & where \ \mathcal{B}_{1,2}(x,y) = 27xy(1-x-y). \end{aligned}$$

Shape functions when d=3:

$$\begin{aligned}\varphi_i(x,y) =& a_1 + a_2x + a_3y + a_4x^2 + a_5y^2 + a_6xy + a_7x^3 + a_8y^3 + \\ &+ a_9x^2y + a_{10}xy^2 + a_{11}\mathcal{B}_{1,3}(x,y) + a_{12}\mathcal{B}_{2,3}(x,y), \quad \forall i = 1,\dots,12\end{aligned}$$

$$\begin{aligned}\textit{where } \mathcal{B}_{1,3}(x,y) &= x^2y(1-x-y)\\ \mathcal{B}_{2,3}(x,y) &= xy^2(1-x-y)\end{aligned}$$

Here below we report the code.

```
function [eval_phi, grad_phi_x, grad_phi_y] =
    shape_functions_masslumping(nodes, qpts)

% Evaluation of mass lumping functions of degree d on
    quadrature points qpts
% Degree:   d = 1 if size(nodes) = 3x2,
%           d = 2 if size(nodes) = 7x2,
%           d = 3 if size(nodes) = 12x2,

if(nargin < 2) % if we want evaluation on the nodes
    qpts = nodes;
end

%example: reference triangle with 6+1 nodes
%   5
%   |  \
%   |    \
%   6      4
%   |  7     \
%   1    2    3
% phi(x,y) = a1 + a2*x + a3*y + a4*x^2 + a5*y^2 + a6*x*y
    + a7*bubble(x,y)
% where bubble(x,y) = 27*x*y*(1 x y)

[n, ~] = size(nodes);
phi = zeros(n,n);

for i=1:n % evaluation of basis coeff.
    phi(:,i) = coeff_basis(nodes,i);
end

S = eval_matrix_masslumping(nodes, qpts);
% for example, if d = 2, > 7 nodes
% here we have the evaluations of 1,x,y,x^2,y^2,xy,bubble
    on the points qpts

eval_phi = S*phi;
```

```
34 % Multipl. of the evaluation matrix S and coeff. matrix
       phi
35 % if qpts=nodes the result will be diagonal
36
37 [grad_phi_x,grad_phi_y] = eval_grad_basis(nodes, qpts,
       phi);
38 %eval of grad of basis functions on qpts
39
40 end
```

At line 8 we can see that, if we do not provide as input also `qpts`, the function will assume `qpts` equal to `nodes`; so, as said before, we want to evaluate the basis functions on the basis nodes.
At line 25 we compute the matrix `phi` that contains the coefficients for the basis (where $\mathcal{L}_i(\mathbf{x}_j) = \delta_{ij}$).
In the other part of the function, we evaluate the basis functions and gradients: let us see in detail the function `eval_grad_basis.m`.

```
1  function [grad_phi_x, grad_phi_y] = eval_grad_basis(nodes
       , qpts, phi)
2
3  % evaluation of Gradient of basis functions on qpts
4
5  if(nargin < 2)
6      qpts = nodes;
7  end
8
9  % for example, if d=2 => 7 nodes
10 % grad phi (x,y) :
11 %(a2 + 2*a4*x + a6*y + a7*y*(1 x y)    a7*x*y)
12 %(a3 + 2*a5*y + a6*x + a7*x*(1 x y)    a7*x*y)
13
14 [n, ~] = size(nodes);
15 [q, ~] = size(qpts);
16
17 grad_phi_x = zeros(q,n);
18 grad_phi_y = zeros(q,n);
19
20 if(n == 3) % degree d = 1
21     for i=1:q
22         for j=1:n
23             grad_phi_x(i,j) = phi(2,j);
24             grad_phi_y(i,j) = phi(3,j);
25         end
26     end
27
28 elseif(n == 7) % degree d = 2
```

```
29      for i=1:q
30      xx = qpts(i,1);
31      yy = qpts(i,2);
32          for j=1:n
33              grad_phi_x(i,j) = phi(2,j) + 2*phi(4,j)*xx +
                    phi(6,j)*yy + ...
34                   phi(7,j)*(yy    yy^2    2*xx*yy);
35              grad_phi_y(i,j) = phi(3,j) + 2*phi(5,j)*yy +
                    phi(6,j)*xx + ...
36                   phi(7,j)*(xx    xx^2    2*xx*yy);
37          end
38      end
39
40  else % n == 12, degree d = 3
41      for i=1:q
42      xx = qpts(i,1);
43      yy = qpts(i,2);
44          for j=1:n
45              grad_phi_x(i,j) = phi(2,j) + 2*phi(4,j)*xx +
                    phi(6,j)*yy + ...
46                   3*phi(7,j)*xx^2 + 2*phi(9,j)*xx*yy + phi
                        (10,j)*yy^2 + ...
47                   phi(11,j)*( xx^2*yy    2*xx*yy*(xx+yy 1))+
                          ...
48                   phi(12,j)*( yy^2*(xx+yy 1)    xx*yy^2);
49              grad_phi_y(i,j) = phi(3,j) + 2*phi(5,j)*yy +
                    phi(6,j)*xx + ...
50                   3*phi(8,j)*yy^2 + phi(9,j)*xx^2 + 2*phi
                        (10,j)*xx*yy + ...
51                   phi(11,j)*( xx^2*(xx+yy 1)    xx^2*yy)+
                          ...
52                   phi(12,j)*( xx*yy^2    2*xx*yy*(xx+yy 1));
53          end
54      end
55  end
56
57  end
```

We highlight in particular the code from line 20, where we make a distinction thanks to the size of `nodes` to recover the degree d, and for each case we compute the gradient of basis function evaluated in the points `qpts` (and we recall another time that they can be equal to the nodes or not).

3 Numerical results

3.1 Convergence analysis

In order to verify the convergence properties of our algorithm, we tested it on the problem

$$\begin{cases} \dfrac{\partial^2 u(x,y,t)}{\partial t^2} - \Delta u(x,y,t) = f(x,y,t) & (x,y,t) \in \Omega \times \mathbb{R}^+ \\ u(x,y,0) = x(1-x)y(1-y) & (x,y) \in \Omega \\ \dfrac{\partial u}{\partial t}(x,y,0) = -x(1-x)y(1-y) & (x,y) \in \Omega \\ u(x,y,t) = xy(1-x)(1-y)e^{-t} & (x,y,t) \in \partial\Omega \times \mathbb{R}^+ \end{cases} \tag{11}$$

where $\Omega = [0,1] \times [0,1]$ and $f(x,y,t) = (2(-x^2+x) + 2(-y^2+y) + (-x^2+x)(-y^2+y))e^{-t}$. The forcing term, the boundary conditions and the initial conditions have been chosen in order to have the exact solution $u(x,y,t) = xy(1-x)(1-y)e^{-t}$ on $\Omega \times \mathbb{R}^+$.

We point out that, for the computation of the errors, a different quadrature rule turned out to be more significant than the mass lumped one: in particular, we used a Dunavant quadrature rule, which was the one adopted for the computation of the error by the original code.

3.1.1 h-convergence

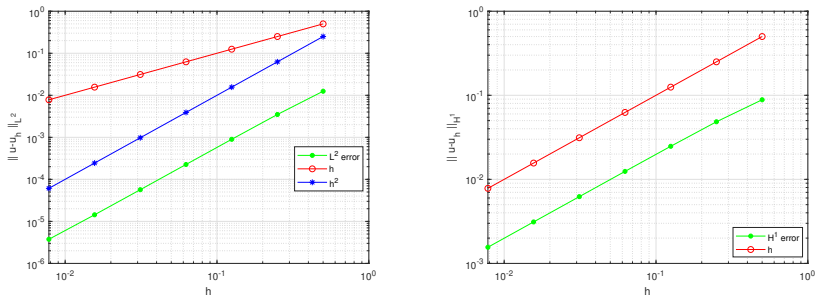
To check the $h-$convergence, i.e. the convergence w.r.t. the size of the mesh, h, we adopt a time step $dt = 1e-5$, a final time $T = 0.2$, and a starting size $h = 0.5$.

From the theory we have that, if u is the real solution of the problem and u_h is the numerical one, then

$$||u - u_h||_{L^2} \sim \mathcal{O}(h^{r+1}) \quad ||u - u_h||_{H^1} \sim \mathcal{O}(h^r)$$

for a sufficiently regular solution u (at least $u \in H^r$).
Our results are consistent with the theoretical results: we expect a second-order convergence using linear basis functions, a third-order convergence with quadratic elements, and a fourth-order convergence adopting cubic elements, and that is what we obtained, at least for the first steps of mesh refinement.

Linear elements



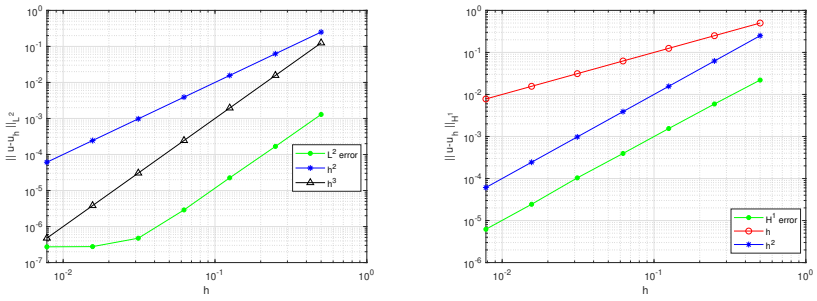
(a) L^2-error (b) H^1-error

Figure 5: Convergence analysis for linear elements

h	L^2-error	L^2-rate	H^1-error	H^1-rate
0.5	1.2477e-02	-	8.8457e-02	-
0.25	3.4930e-03	1.8368	4.8401e-02	0.8699
0.125	8.9693e-04	1.9614	2.4734e-02	0.9685
0.0625	2.2586e-04	1.9895	1.2434e-02	0.9922
0.0313	5.6719e-05	1.9935	6.2254e-03	0.9981
0.0156	1.4347e-05	1.9831	3.1137e-03	0.9995
0.0078	3.7515e-06	1.9352	1.5570e-03	0.9999

Table 1: Errors for linear FE

Quadratic elements



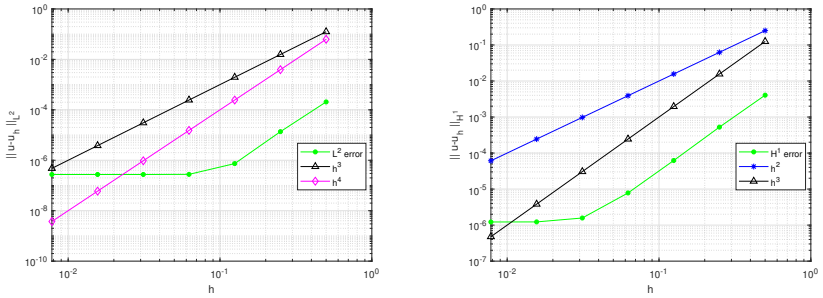
(a) L^2-error (b) H^1-error

Figure 6: Convergence analysis for quadratic elements

h	L^2-error	L^2-rate	H^1-error	H^1-rate
0.5	1.2921e-03	-	2.1995e-02	-
0.25	1.6758e-04	2.9468	5.9279e-03	1.8916
0.125	2.2514e-05	2.8960	1.5449e-03	1.9400
0.0625	2.8924e-06	2.9605	3.9438e-04	1.9699
0.0313	4.7475e-07	2.6071	1.0356e-04	1.9290
0.0156	2.7747e-07	0.7748	2.4372e-05	2.0872
0.0078	2.7348e-07	0.0209	6.2054e-06	1.9736

Table 2: Errors for quadratic FE

Cubic elements



(a) L^2-error (b) H^1-error

Figure 7: Convergence analysis for cubic elements

h	L^2-error	L^2-rate	H^1-error	H^1-rate
0.5	2.0568e-04	-	4.0431e-03	-
0.25	1.3620e-05	3.9166	5.2500e-04	2.9451
0.125	7.3797e-07	4.2060	6.2136e-05	3.0788
0.0625	2.7613e-07	1.4182	7.8114e-06	2.9918
0.0313	2.7305e-07	0.0162	1.5749e-06	2.3103
0.0156	2.7305e-07	0.0000	1.2269e-06	0.3602
0.0078	2.7305e-07	0.0000	1.2212e-06	0.0067

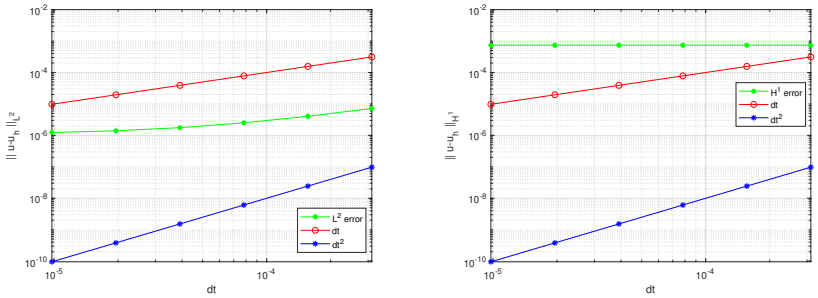
Table 3: Errors for cubic FE

We observe that the plateau is reached earlier if the degree of the finite elements is higher; that happens because the part of the error due to the time discretization is more significant with higher order, where the space error is smaller.
Moreover, we can see that the H^1-error, despite a lower convergence rate, seems to keep its convergence order longer than the L^2-error, i.e. in the H^1 case the plateau appears later than in the L^2 case.

3.1.2 dt-convergence

To check the $dt-$convergence, i.e. the convergence w.r.t. the time step, dt, we adopt a mesh size $h = 0.02$, a final time $T = 0.5$, and a starting time step $dt = 0.00125$, in the case of degree $d = 3$. If we want to check the convergence also for degree $d = 1$ or $d = 2$ we have to change these parameters, because with lower degree in space the error is dominated by the spatial error, and we do not see the expected convergence.
For example we can decrease the size h, and consequently also the starting dt, in order to satisfy the CFL condition: we can see good results for $d = 2$ with

$h = 0.01$ and $dt = 0.000625$, and for $d = 1$ with $h = 0.005$ and $dt = 0.0003125$ (only in L^2-error, because for H^1 the space part of the error is so big that we cannot see any convergence).

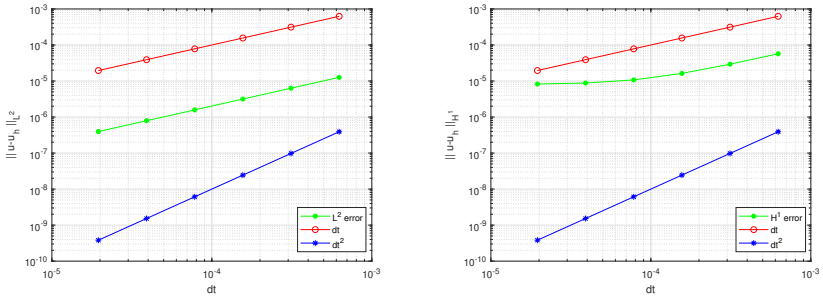


(a) L^2-error (b) H^1-error

Figure 8: Convergence analysis for linear elements

dt	L^2-error	L^2-rate	H^1-error	H^1-rate
3.125e-04	7.190e-06	-	7.3871e-04	-
1.5625e-04	4.056e-06	0.8260	7.3833e-04	0.00072
7.8125e-05	2.511e-06	0.6917	7.3833e-04	0.00016
3.90625e-05	1.762e-06	0.5112	7.3832e-04	0.00003
1.95531e-05	1.404e-06	0.3276	7.3824e-04	0.0000
9.7656e-06	1.233e-06	0.1868	7.3824e-04	0.0000

Table 4: Errors for linear FE

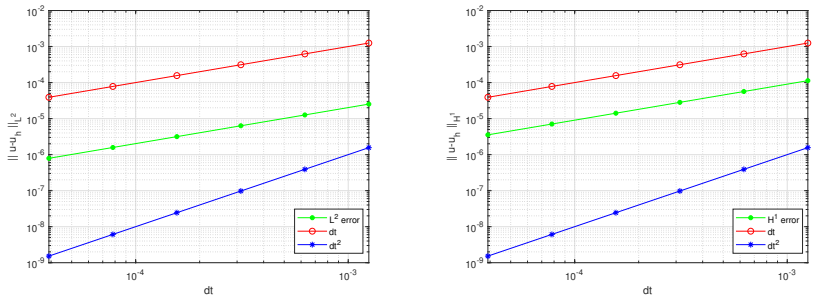


(a) L^2-error (b) H^1-error

Figure 9: Convergence analysis for quadratic elements

dt	L^2-error	L^2-rate	H^1-error	H^1-rate
6.25e-04	1.2632e-05	-	5.6975e-05	-
3.125e-04	6.3171e-06	0.9997	2.9309e-05	0.9590
1.5625e-04	3.1589e-06	0.9998	1.6263e-05	0.8497
7.8125e-05	1.5797e-06	0.9995	1.0722e-05	0.6011
3.90625e-05	7.9008e-07	0.9979	8.8009e-06	0.2848
1.9553125e-05	3.9563e-05	-	8.2499e-06	0.0933

Table 5: Errors for quadratic FE



(a) L^2-error (b) H^1-error

Figure 10: Convergence analysis for cubic elements

dt	L^2-error	L^2-rate	H^1-error	H^1-rate
1.25e-03	2.5255e-05	-	1.1294e-04	-
6.25e-04	1.2632e-05	0.9995	5.6492e-05	0.9995
3.125e-04	6.3170e-06	0.9998	2.8251e-05	0.9997
1.5625e-04	3.1588e-06	0.9999	1.4128e-05	0.9998
7.8125e-05	1.5795e-06	0.9999	7.0664e-06	0.9995
3.90625e-05	7.8978e-07	0.9999	3.5378e-06	0.9981

Table 6: Errors for cubic FE

From the theory we have that, if u is the real solution of the problem and u_h is the numerical one, then

$$||u - u_h||_{L^2} \sim \mathcal{O}(dt) \quad ||u - u_h||_{H^1} \sim \mathcal{O}(dt).$$

Our results are consistent with the theoretical results: we expect a linear convergence for every choice of the degree of the basis functions, since we are using a first order approximation for the time derivative, and this is what we obtain, before the plateau appears.

3.2 Comparison with the non mass lumping method

We can compare our method to a classic Finite Element in terms of error, but also in terms of computational time and condition number for the matrix M, that, we recall here, we need to invert at every time iteration.
We can expect that our method outperforms the traditional Finite Element in terms of time needed to compute the solution, but also for the condition number, that we can expect to be of order 1 since M is diagonal, while in the traditional case the condition number will increase with the number of element used.
For the comparison we use the previous example with $d = 1$ and a starting $h = 0.25$, with 5 iteration (in terms of elements used we have $n_{els} = [8, 32, 128, 512, 2048]$). In terms of error, we can expect that our ML method has the same behaviour as the traditional method, since, even if it could be less accurate, the traditional method has a bad conditioning for the mass matrix that can lead to a worse accuracy.

3.2.1 Computational time

As we can see in Fig. 11, the computational time in ML algorithm grows from 0.5 to 1.5 sec, while in the traditional case it grows from 0.7 to 25 sec. We observe a big difference in the computational time while increasing the number of elements used.

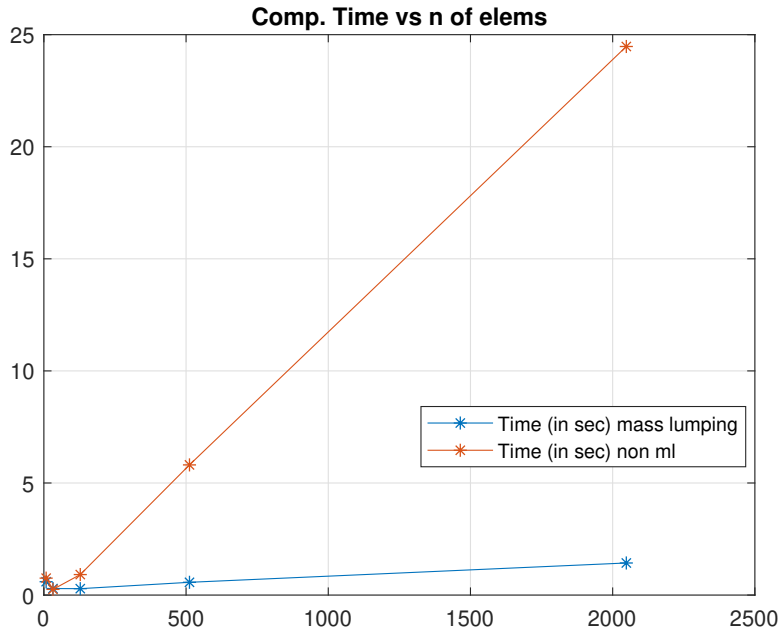


Figure 11: Computational time vs Number of elements used, $d = 1$

3.2.2 Condition number of the mass matrix

The difference in terms of conditioning number for the mass matrix is of course in favour of the mass-lumping method, where $cond(M)$ is more or less equal to 1 for every iteration, while in the second case it rapidly increases with the number of elements used, reaching a value of 955 in the last iteration.

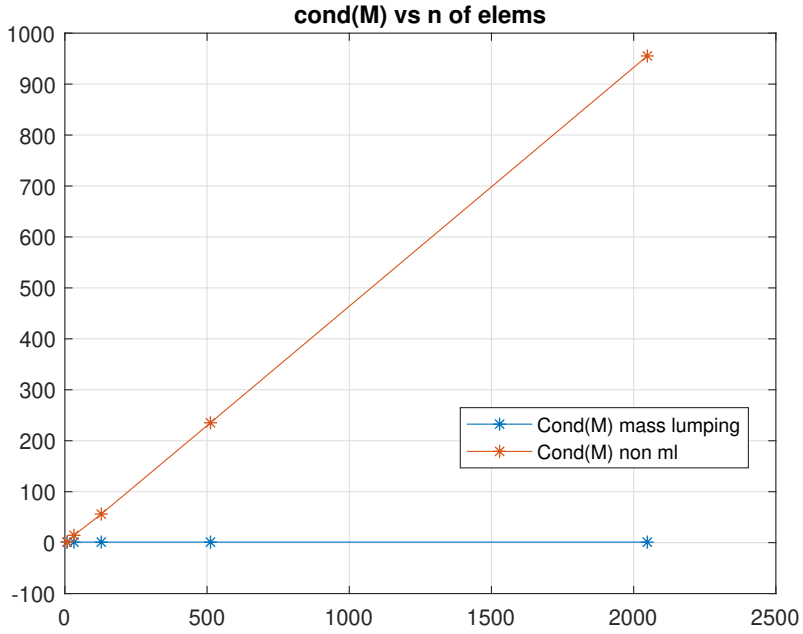


Figure 12: Condition number of matrix M vs Number of elements used, $d = 1$

3.2.3 Error accuracy

As we expect, both methods have a second order convergence, and we observe that the error for the mass lumping method is always less than the error in the traditional case, probably due to the bad conditioning of the matrix M.

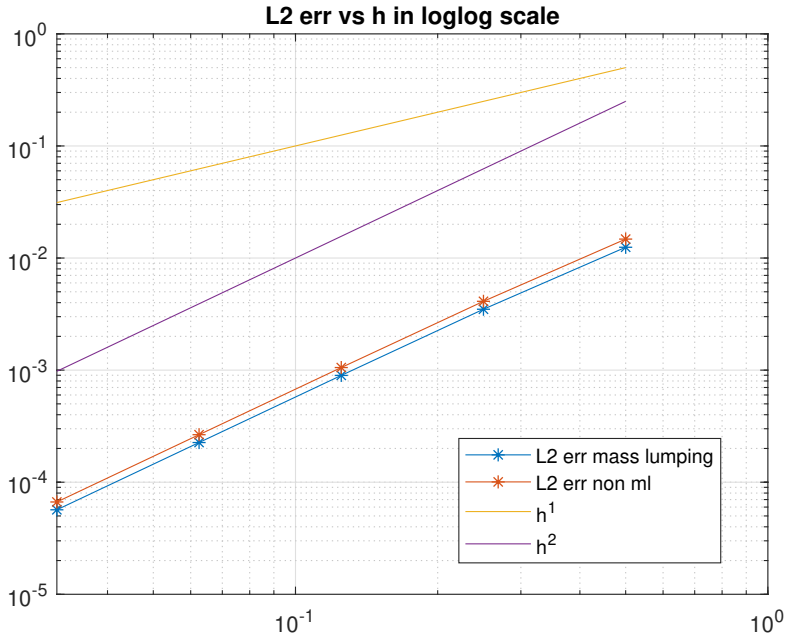


Figure 13: L^2 error vs h in loglog scale, $d = 1$

3.3 Some interesting cases

After the numerical tests with the aim to verify the convergence and accuracy properties of our code, we tested the algorithm on some numerical experiments, because we were interested in observing the behaviour of the solutions in more complex cases than the ones used in the numerical tests. In particular, we focused on two main features of the problem: a non uniform velocity and a complex geometry of the domain.
The main aspect of the problems we consider in this section is that it is not available an analytical solution with which one can compare the numerical one: in fact, these experiments were not used to compute errors or convergence rates. The only thing one can do is try to interpret physically the qualitative behaviour of the wave, and at most check the stability of the solution (i.e., the norm of the solution does not grow up too fast in time).

3.3.1 The paper example

This first attempt with a problem with no analytical solution is contained in [1].

We model the wave problem in the domain $\Omega = (0, 12) \times (0, 12)$ with homogeneous initial and boundary conditions and $k(\mathbf{x}) = 1$; the forcing term is the

product of a spatial and a temporal part, $f(\mathbf{x}, t) = f_1(t)\, f_2(\mathbf{x})$, with

$$f_1(t) = \begin{cases} 2a(2a(t-b)^2 - 1)exp(-a(t-b)^2) & 0 \leq t \leq 3.5 \\ 0 & otherwise \end{cases}$$

and

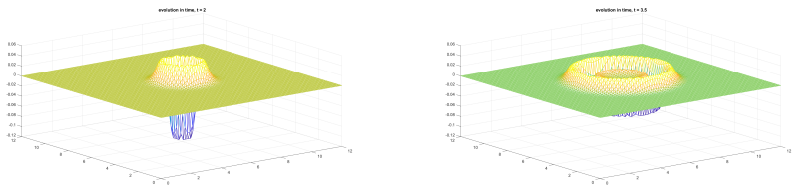
$$f_2(\mathbf{x}) = exp(-7|\mathbf{x} - \mathbf{x}_S|)\,.$$

The parameters are $a = (\frac{\pi}{1.31})^2$, $b = 1.35$ and $\mathbf{x}_S = (6, 6)$. In poor words: the source is quasi-punctual, located at the center of the domain and also effective for a limited time.

What we expect in this situation is a solution which is symmetric with respect to the axes of symmetry of the square domain. Moreover, what one can qualitative observe is the interference phenomenon, in which two wave fronts interact for an instant, in a constructive or destructive way, and then proceed unchanged (superposition principle).

We simulate the behaviour of the wave until the time $T = 20\, s$ with a time step $dt = 0.0001\, s$, saving the snapshots every $0.5\, s$ and using quadratic elements. The size of the mesh is $h = 0.16$.

Initially, we see how the initial quasi-punctual pulse generates a single circular wave front, which grows in time until it reaches the walls of the domain.



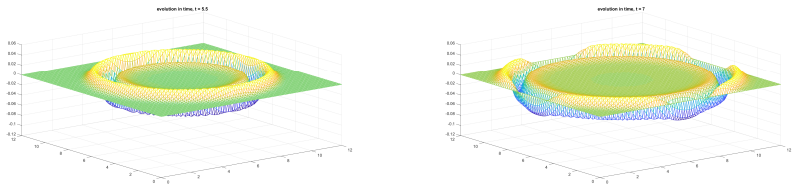
(a) Time = 2 (b) Time = 3.5

Figure 14: Plot of the wave before hitting the wall

At about $t = 5.5\, s$, the circular wave front hits all the four edges at the same time, and at $t = 7\, s$ it starts to be reflected, due to the homogeneous Dirichlet boundary conditions, generating four clearly distinct wave fronts. We observe that the original circular wave front does not bounce back entirely at the same time: in particular, when the reflection begins, some parts of the front still have to reach the corners of the domain.

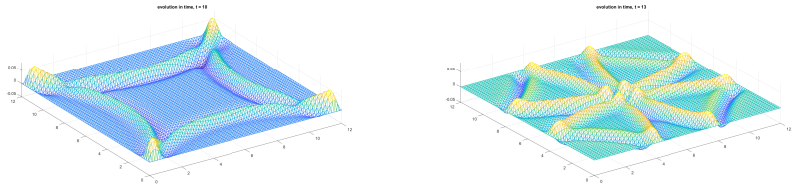
At $t = 10\, s$ the original wave front is completely reflected, and for subsequent instants we observe the formation of four more wave fronts.

At this point, the eight wave fronts run through the domain, until, for $t = 20\, s$, we reach a situation which resembles the one at $t = 10\, s$, but is much more



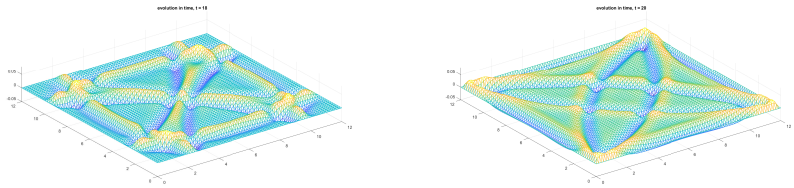
(a) Time = 5.5 (b) Time = 7

Figure 15: The wave is reflected



(a) Time = 10 (b) Time = 13

Figure 16: More wave fronts are generated



(a) Time = 18 (b) Time = 20

Figure 17: Another loop begins

complex, due to the greater number of wave fronts, which can be now barely identified.

The interaction of the wave fronts with the corners is probably the cause of this phenomenon. For this reason we expect the situation to become much more complex every time that the waves reach the boundary.

This experiment is interesting to test the *dispersion properties* of the algorithm on long times (this was actually the goal of this numerical experiment in the original paper).

3.3.2 Non uniform velocity

In this experiment we try to repeat a numerical example from the paper [5].

Let us consider a case in which the density of the medium is not constant along the domain: the numerical consequence of this physical fact is that the function k is no more constant.
Define $\Omega = (0,3) \times (0,2)$ and

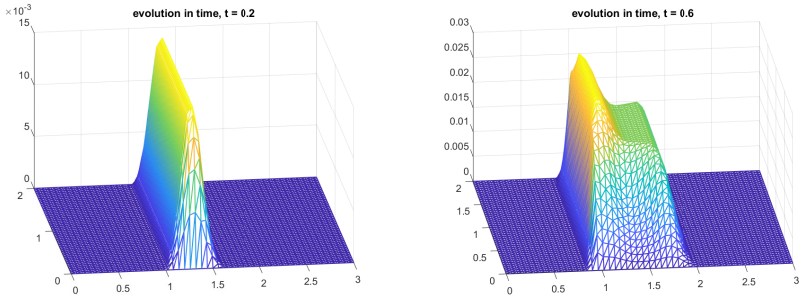
$$k(x,y) = \begin{cases} 1 & x \geq 1 \\ 0.1 & otherwise \end{cases},$$

with homogeneous initial and boundary conditions, and

$$f(x,y,t) = \begin{cases} 1 & 1.2 \leq x \leq 1.4,\ t \leq 0.2 \\ 0 & otherwise \end{cases}.$$

This is basically the model of a non-homogeneous plate. We notice that, being this a model for mechanical waves, the region where the propagation velocity is higher is also the more dense one.
We expect a different behaviour of the wave in the two different regions of the plate: in particular, we expect the wave travelling in the right part of the plate to be much faster than the one travelling in the left part.
We simulate the behaviour of the wave until the time $T = 8\,s$ with a time step $dt = 0.0001\,s$, saving the snapshots every $0.2\,s$ and using cubic elements. The size of the mesh is $h = 0.06$.

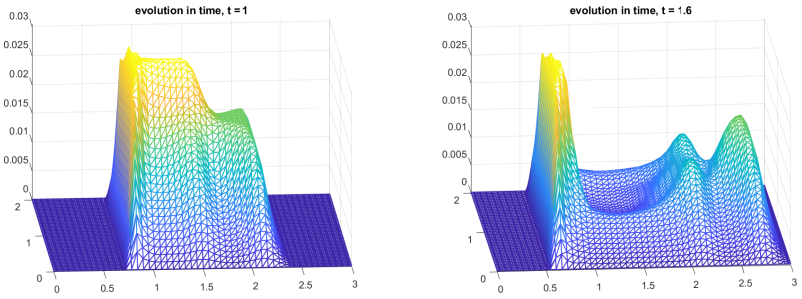
The initial excitation splits into two planar wave fronts travelling in the two opposite directions, and the solution is symmetric until $t = 0.2\,s$, when the left wave front hits the discontinuity boundary.



(a) Time = 0.2 (b) Time = 0.6

Figure 18: Plot of the wave at initial times

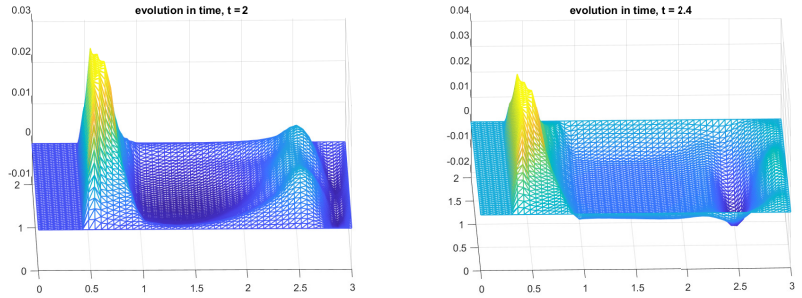
Advancing in time, the difference between the wave propagation velocities in the two different media is evident. Moreover, we observe that just after the instant when the left wave front enters the less dense region, due to the discontinuity surface, a new right-travelling wave front is created. After that, the first right wave front hits the right wall at $t = 1.6\,s$ and is reflected.



(a) Time = 1 (b) Time = 1.6

Figure 19: Plot of the wave before hitting the walls

Afterwards, the two wave fronts in the right region meet and pass one through each other, and at about $t = 2.4\,s$ also the second right-travelling wave front hits the boundary.



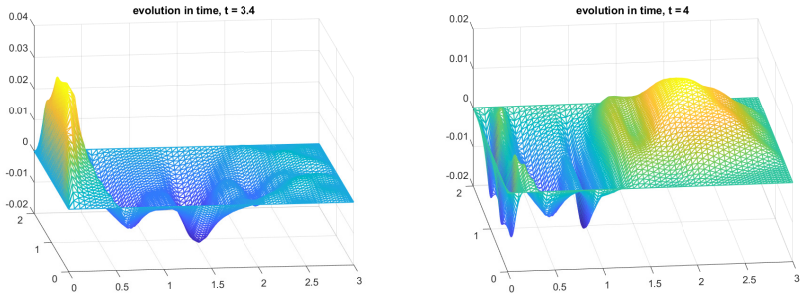
(a) Time = 2 (b) Time = 2.4

Figure 20: Interaction and second hit

The last clearly identifiable significant instant is when the left wave front finally hits the left boundary, at $t = 3.4\,s$, but the discontinuity of the density in the material generates multiple reflections, which interact with each other at later times: even for $t = 4\,s$ it is almost impossible to detect every single wave front which was earlier created and to understand the origin of the peculiar shapes of the of the different waves.

Anyway, we note two quantitative aspects: the solution is stable at least until our final time $T = 8\,s$; from the advancing of the wave fronts towards the right and the left directions before hitting the walls we are able to compute the velocities in the two different regions, which agree with the ones computed from k.

We also qualitative observe a general good match between our snapshots and the ones reported in the paper.



(a) Time = 3.4 (b) Time = 4

Figure 21: Left hit and multiple reflections

3.3.3 Some particular domains

The teardrop hole This first experiment with a complex geometry is inspired by [1]. We treat here a problem of wave propagation in an exterior domain.

The domain is the complement of a "cone-sphere"-shaped obstacle, and we consider again homogeneous initial and boundary conditions and velocity 1. The source, similar to the one adopted in Section 3.3.1, is a point located at the left of the obstacle.

The interesting behaviour to observe in this experiment is the diffraction of the incident wave caused by the presence of the obstacle. We see in particular the diffraction phenomenon due to the summit of the cone.

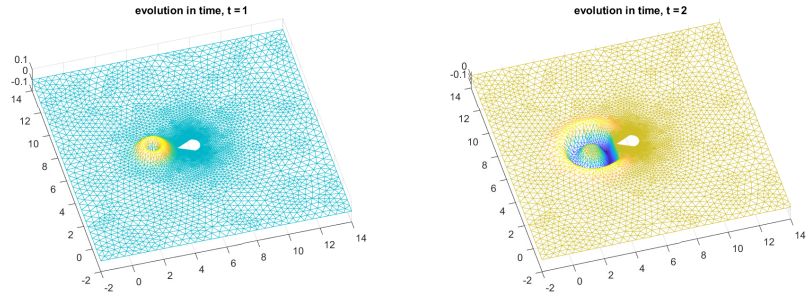
We simulate the behaviour of the wave until the time $T = 7\,s$ with a time step $dt = 0.0001\,s$, saving the snapshots every $0.2\,s$ and using linear elements. The size of the mesh is about $h = 0.01$.

For initial times, we observe the classic evolution of the quasi-punctual source, until the wave reach the top of the drop, at about $t = 2\,s$.

Then, we can clearly appreciate the diffraction phenomenon.

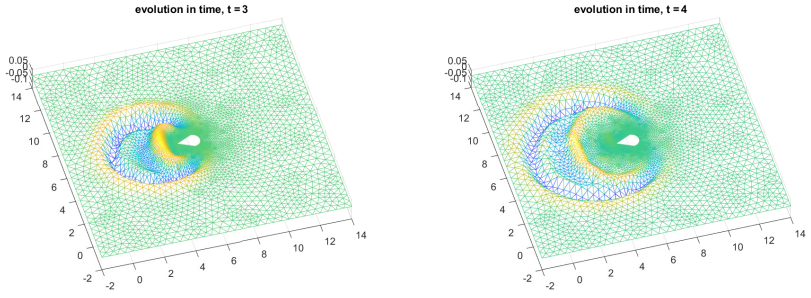
At $t = 5\,s$ the first wave front hits the wall in front of the drop. Unfortunately, for subsequent times we observe the generation of some sort of local instabilities: the wave fronts are no more clear, and we can barely identify them.

The most complex issue to handle to perform this experiment was the geometry. First of all, it has been quite difficult to design the shape of the hole in the middle of the domain. Moreover, there is a problem with the mesh generation done by `pdetool`: the whole teardrop is the result of a series of set operations, such that union and difference, done by the software on some starting "elementary" geometric figures, such as rectangles and circles. The problem is that the



(a) Time = 1 (b) Time = 2

Figure 22: Initial times



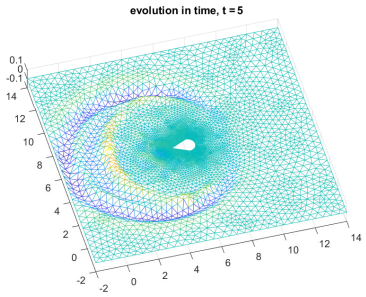
(a) Time = 3 (b) Time = 4

Figure 23: The diffraction

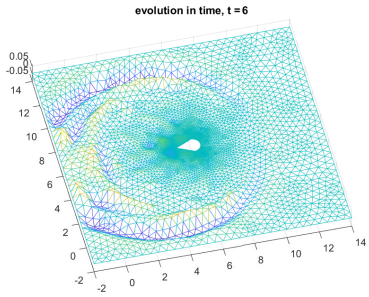
mesher, during the mesh initialization, considers every figure as a single entity, and so there are some limitations in the mesh design (for example, a triangle of the mesh cannot cross the virtual edge of one of the original figures). This issue is exasperated by the fact that the figures are not glued together perfectly, so very small triangles are needed to tessellate these imperfections. The result of all of this is a mesh with a uselessly small size and with very non-uniform size of the triangles.

The narrow channel This numerical experiment is inspired by the example contained in [2]. This is again an attempt of handling complex geometries, but now it is a wave propagation problem in an interior domain.
In particular the domain consists of two rectangles, of dimensions (4 x 4) and (4 x 1.5), connected by a narrow channel, (0.02 x 1).

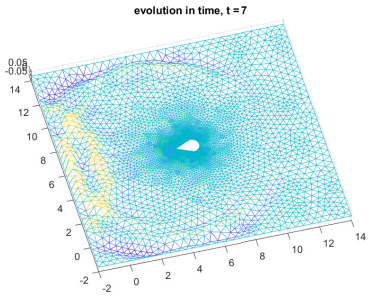
As in the previous experiment, the interesting behaviour to observe in this experiment is the diffraction of the wave caused by the channel in the smaller rectangle.



(a) Time = 5



(b) Time = 6



(c) Time = 7

Figure 24: Generation of instabilities

We impose homogeneous Dirichlet boundary and initial conditions, and the forcing term is again a quasi-punctual source, concentrated in space and time. It is centered in the bigger rectangle, and closer to the channel than to the other wall. The velocity is $k = 3$.

We simulate the behaviour of the wave until the time $T = 3\,s$ with a time step $dt = 0.000001\,s$, saving the snapshots every $0.2\,s$ and using linear elements. The size of the mesh is $h = 0.025$.

At initial times, as usual, we observe the standard evolution of a quasi-punctual source, until the first wave-front hits the wall, at $t = 0.8\,s$. Then it can be clearly seen that a part of the wave enters the channel, while the other part is reflected by the homogeneous Dirichlet conditions. At $t = 1.6\,s$ the wave reaches the end of the channel (Figure 28).

Figure 25: Computational domain and mesh

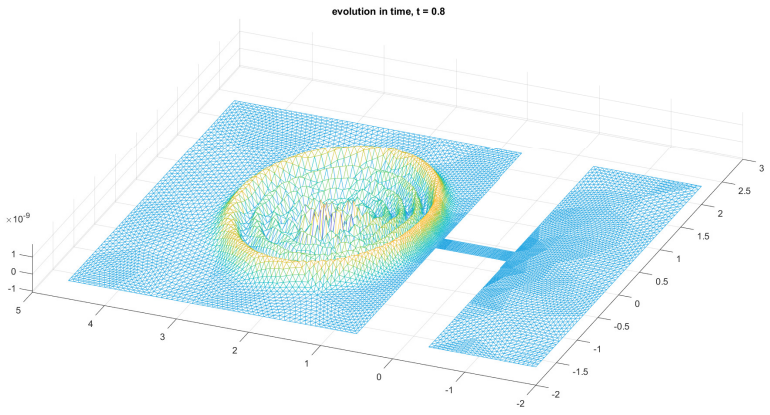


Figure 26: Time = 0.8

Immediately after, the diffraction phenomenon begins, as we can see in Figure 30. Its causes are well-known: the channel works as a concentrated source for the wave that ran across it, and the generation of circular wave fronts happens in the smaller rectangle. The situation in the bigger rectangle, where the

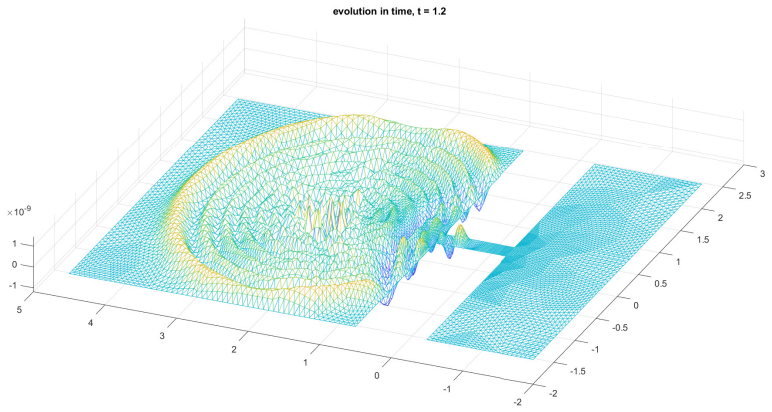


Figure 27: Time = 1.2

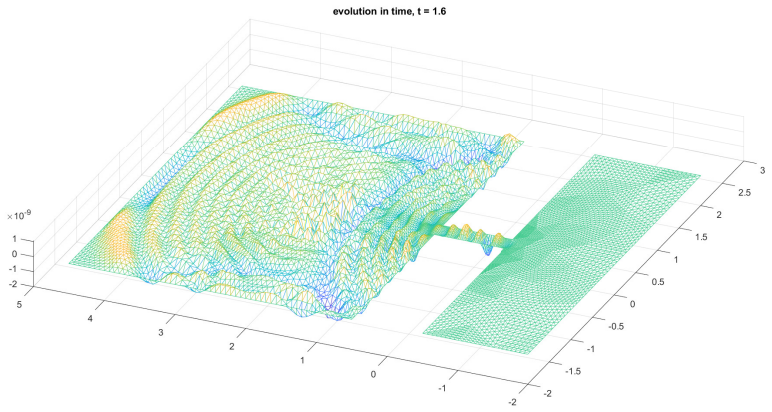


Figure 28: Time = 1.6

initial source was located, is very messy, and after $t = 1.6\,s$, due to the multiple reflections produced by the homogeneous Dirichlet conditions, it is completely impossible to interpret what is happening.

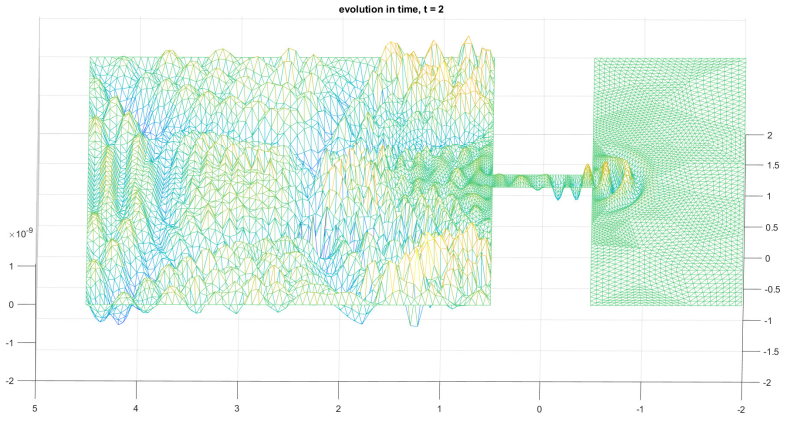


Figure 29: Time = 2

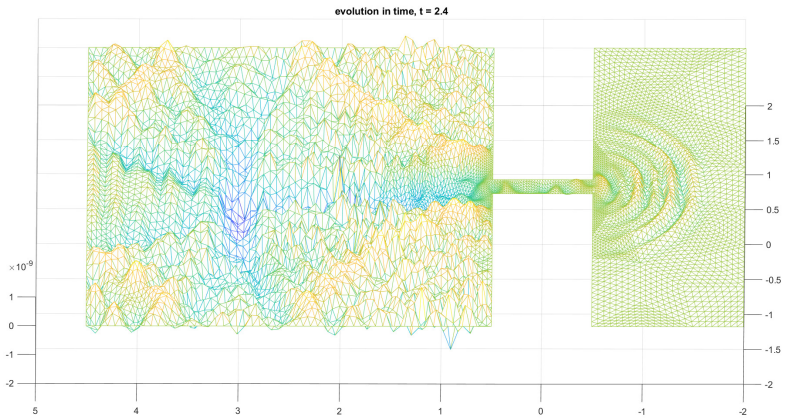


Figure 30: Time = 2.4

4 Conclusions

4.1 Comments on the numerical results

As we expect, the ML algorithm is much less expensive in terms of computational time than the standard FE method, thanks especially to the trivial inversion of the mass matrix, even if the number of degrees of freedom in the ML case is bigger (and so the mass matrix is bigger too).
The fact that was not trivial was keeping the accuracy in the error analysis: nonetheless, we observed that the error is not only comparable with the FE algorithm, but even smaller. Therefore, we conclude that the less accuracy of

the quadrature rule is more than compensated by the very bad conditioning of the mass matrix in the standard case.
Hence, at this point we can assert that this ML method is better than the classic implementations of the FE method; nonetheless, for a better understanding of the behaviour of this algorithm w.r.t. other schemes, and for a more complete comparison, we should consider also the results of other methods, such as other mass-lumping techniques, FE implementations with preconditioning features, or even SE methods.

4.2 Further developments

Some of the most immediate developments concerning this project can be:

- implement a higher order approximation of the time derivative, or use an unconditionally stable method;
- generalize the code to implement the Mass Lumping method with finite elements of higher order in space;
- generalize the algorithm in order to have the possibility to consider fully Neumann, Robin or mixed problems;
- consider the case of the vectorial wave equation, i.e. where the unknown field $\mathbf{u}$ is a vector;
- consider the case of $\Omega \subseteq \mathbb{R}^3$, e.g. to model in a better way the propagation of earthquakes (actually, cfr [1], this is not as straightforward as one might think);
- improve the handling of the mesh for complex domains, for example to have the possibility to use (non-)uniform or (non-)structured meshes.

Some further extensions of this work may be:

- consider the case of anisotropic diffusion, i.e. consider a second order tensor, $\mathbf{A} = \mathbf{A}(\mathbf{x}, t)$, instead of the scalar k;
- include an advection term, for example $\mathbf{b}(\mathbf{x}, t) \cdot \nabla u(\mathbf{x}, t)$, or a reaction term, $a(\mathbf{x}, t)\, u(\mathbf{x}, t)$;

References

[1] G. Cohen et al. “Higher order triangular FE with mass lumping for the wave equation”. In: *SIAM Journal on Numerical Analysis* 38.6 (2001), pp. 2047–2078.

[2] J. Diaz and M. J. Grote. “Energy conserving explicit local time stepping for second-order wave equations”. In: *SIAM Journal on Scientific Computing* 31.3 (2009), pp. 1985–2014.

[3] S. Geevers, W. A. Mulder, and J. J. W. van der Vegt. “New higher-order mass-lumped tetrahedral elements for wave propagation modelling”. In: (2018).

[4] M. S. Gockenbach. *Understanding and Implementing the Finite Element Method*. SIAM, 2006.

[5] M. J. Grote, A. Schneebeli, and D. Schotzau. “Discontinuous Galerkin FEM for the wave equation”. In: *SIAM Journal on Numerical Analysis* 44.6 (2006), pp. 2408–2431.

[6] J. L. Lions and E. Magenes. *Non-homogeneous boundary value problems and applications*. Springer Verlag, 2012.

[7] MathWorks. *Geometry and Mesh*. 2020. URL: `https://it.mathworks.com/help/pde/geometry-and-mesh.html?s_tid=CRUX_lftnav` (visited on 02/13/2020).

[8] S. Salsa. *Equazioni a derivate parziali. Metodi, modelli e applicazioni*. Springer Verlag, 2012.