DiffusionConvectionReaction2d

Simone Chiominto

January 2023

Contents

Chapter 1

Introduction

DiffusionConvectionReaction2d is a repository containing Matlab code for the numerical solution of Diffusion Convection Reaction problems in a 2D-polygonal domain. It also provides classes and methods for solving simple Parabolic problems on a 2D-polygonal domain.

The code exploits heavily OOP in Matlab which, at the cost of little longer computation time helps to make a more structured code. We also exploited the possibility to use *handle classes* which are the only way to simulate the behavior of pointers in Matlab without resorting to global variables.

The main useful classes for an external user are `ApproxDiffusionConvectionProblem2D` and `ApproxParabolicProblem` which contain the methods for solving numerically PDE using respectively Galerkin (and Petrov-Galerkin) method and the method of lines with Crank-Nicolson method for the numerical solution of ODE. The classes `DiffusionConvectionProblem2D` and `ParabolicProblem` provide an implementation of an abstract elliptic and parabolic problem. The class `Mesh` is used mostly to simulate the behavior of pointers in Matlab inserting the heavy structures coming as output from the Triangler in an Handle Class. The class `Element` should not concern an external user, but a description of his functioning is anyway provided in this documentation.

Along this documentation we will often refers to the Barbera's Triangler documentation that can be found at
`/Classes/@Mesh/Triangolatore/Long/bbtr30/attachments/User's handbook.pdf`

1.1 Usage Example

Here we provide a simple example for the use of this repository for the solution of an elliptic problem. The

```
addpath("Classes")

%problem parameters
mu=1;
beta=[0;0];
```

```
gamma=0;
f=@(x)32*(x(1,:).*(1-x(1,:))+x(2,:).*(1-x(2,:)));

%description of the domain and boundary conditions
Domain =%see Trinagler Documentation
BC =%see Triangler Documentation
boundaryFunctions{1}= @(x) zeros(size(x(1,:)));

%exact solution
u=  @(x)16*x(1,:).*(1-x(1,:)).*x(2,:).*(1-x(2,:));
grad_u =  @(x)  [16*(1-2*x(1,:)).*x(2,:).*(1-x(2,:));...
                 16*x(1,:).*(1-x(1,:)).*(1-2*x(2,:))];

%create instance of the problem
problem=DiffusionConvectionReactionProblem2D...
                (mu,beta,gamma,f,...
                 Domain,BC,boundaryFunctions,...
                 u,grad_u);

RefiningOptions= %see Triangler Documentation
RefiningOptions.AreaValue  = 0.1; %max Area of the triangles
RefiningOptions.AngleValue = 20; %min angle of the triangles

%load presaved P1 and P2 reference elements
load("refElements.mat")

%create mesh using the Triangler
mesh=Mesh(problem.domain,problem.BC,RefiningOptions);

%create object describing the approximate problem
approx_problem=ApproxDiffusionConvectionReactionProblem2D...
                (problem,mesh,courantEl);
%compute stiffness matrix and forcing vector
approx_problem.generateLinearSystem...
                ("correctConvection",false,"massLumping",false);
%solve the problem
approx_problem.solve()
%plot solution
approx_problem.plot()
```

Chapter 2

Classes

In this chapter we provide a description of the classes and their properties and methods.

2.1 Element

`Element` is a class implementing a triangle (real element) used for the numerical solution of a two-dimensional Diffusion Reaction Convection problem and two-dimensional Parabolic Problem. We call reference triangle $\hat{T}$ the triangle with vertices $(1,0)$, $(0,1)$, $(0,0)$. A real element E is obtained from the reference triangle through an affine transformation $F_E : \mathbb{R}^2 \to \mathbb{R}^2$ defined as

$$F_E(\hat{\mathbf{x}}) = \mathbf{a} + \mathbf{B}\hat{\mathbf{x}} \tag{2.1}$$

2.1.1 Properties

coordinates

This property is a 3×2 array containing the coordinates of the three vertices of the triangle given counterclockiwise .

B

This property is the 2×2 matrix $\mathbf{B}$ in the affine transformation F_E in (2.1).

Binv

This property is the matrix inverse of $\mathbf{B}$.

phi

This property is a length `nDof` (see 2.1.1) cell array whose components are function handles. The i-th component is the basis function ϕ_i of the element.

gradPhi

This property is a length `nDof` (see 2.1.1) cell array whose components are function handles. The i-th component is the gradient $\nabla\phi_i$ of the basis function ϕ_i of the element.

hessPhi

This property is a length `nDof` (see 2.1.1) cell array whose components are function handles. The i-th component is the Hessian matrix H_{ϕ_i} of the basis function ϕ_i of the element.

Fe

This property is a function handle implementing the affine transformation F_E defined by $F_E(\hat{\mathbf{x}}) = \mathbf{a} + \mathbf{B}\hat{\mathbf{x}}$.

FeInv

This property is a function handle implementing the inverse of affine transformation F_E defined as $F_E^{-1}(\mathbf{x}) = \mathbf{B}^{-1}(\mathbf{x} - \mathbf{a})$

Area

This property is the measure of the area of the triangle E.

nDoF

This property is the number of degree of freedom of the element E. This number is equal to 3 for the $\mathbb{P}_1$ elements and 6 for the $\mathbb{P}_2$ elements.

type

This property is a string whose value is either "P1" or "P2".

2.1.2 Methods

Element()

Constructor of the class `Element`.
Parameters:

- `coordinates`: 3×2 array containing the coordinates of the three vertices of the triangle given counterclockiwise .
- `refElement`: struct describing the reference element used. It needs to have 3 fields: `refElement.phi` is a length `nDoF`-length array with the basis function of the element, `refElement.gradPhi` is a length `nDoF`-length array with the gradients of the basis function of the element and

`refElement.hessPhi` is a length `nDoF`-length array with the hessians of the basis function of the element

Returns:

- `el`: object of class `Element`

getMaxLength()

Returns the maximum length of the three sides of the element.
Parameters:

- obj: object of class `Element`

Return:

- `h_e`: maximum length of the three sides of the element.

getdx()

Returns a lenght-3 array with values $(x_3 - x_2, x_1 - x_3, x_2 - x_1)$ where x_i is the x-coordinate of the i-th vertex
Parameters:

- `obj`: object of class `Element`

Return:

- `dx`: array with values $(x_3 - x_2, x_1 - x_3, x_2 - x_1)$ where x_i is the x-coordinate of the i-th vertex.

getdy()

Returns a lenght-3 array with values $(y_2 - y_3, y_3 - y_1, y_1 - y_2)$ where y_i is the y-coordinate of the i-th vertex
Parameters:

- `obj`: object of class `Element`

Return:

- `dy`: array with values $(y_2 - y_3, y_3 - y_1, y_1 - y_2)$ where y_i is the y-coordinate of the i-th vertex.

2.2 Mesh

`Mesh` is a handle class (pointer in Matlab) implementing a triangulation of a 2d polygonal domain. The construction of this class uses the Barbera's Trianglers which is contained in the folder `@mesh/triangolatore`.

2.2.1 Properties

geom

This property is a struct returned by the the Barbera's Triangler and it is described in its User's Handbook

areaMax

This property is the maximum value of the areas of the triangles in the mesh

diamMax

This property is the maximum length of the sides of the triangles in the mesh

nDegreeOfFreedom

This property is the number of degree of freedom (which are not subject to Dirichlet conditions) of the mesh

elements

This property is an array of object of class `Element` (see 2.1). Contains elements corresponding to all the triangles of the triangulation.

2.2.2 Methods

Mesh()

Constructor of class `Mesh`.
Parameters:

- `Domain`: struct containing the description of the domain as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `BC`: struct containing the description of the boundary conditions of the problem as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `RefiningOptions`: struct containing information on the discretization requested by the user as requested by the the Barbera's Triangler. It is described in its User's Handbook.

Return:

- `obj`: object of class `Mesh`.

P2()

Modifies the structures in `Mesh.geom` to deal with $\mathbb{P}_2$ elements.
Parameters:

- `obj`: object of class `Mesh`

getAreaMax()

Returns the value stored in `Mesh.areaMax`. If it is empty, it computes the maximum value of the areas of the triangles in the mesh e stores it in `Mesh.areaMax`
Parameters:

- `obj`: object of class `Mesh`

Return:

- `area`: maximum value of the areas of the triangles in the mesh

getDiamMax()

Returns the value stored in `Mesh.diamMax`. If it is empty, it computes the maximum length of the sides of the triangles in the mesh e stores it in `Mesh.diamMax`
Parameters:

- `obj`: object of class `Mesh`

Return:

- `h`: maximum length of the sides of the triangles in the mesh

getDoF()

Returns the value stored in `Mesh.nDegreeOfFreedom`. If it is empty, it computes the number of degree of freedom (which are not subject to Dirichlet conditions) of the mesh and stores it in `Mesh.nDegreeOfFreedom`
Parameters:

- `obj`: object of class Mesh

Return:

- `nDoF`: number of degree of freedom of the mesh

getEl()

Returns the e-th component of the array stored in `Mesh.elements`. If it is empty, creates an object of class Element correspondent to the e-th triangle in `Mesh.geom` e stores it in the e-th component of `Mesh.elements`
Parameters:

- `obj`: object of class mesh

- `refElement`: description of a reference element as explained in 2.1.2.
- `e` (int): index of the triangle in `Mesh.geom` for which we want to create an object of class `Element`

Return:

- `el`: object of class Element

delEl()

Deletes the property `Mesh.elements` from the `Mesh` object
Parameters:

- `obj`: object of class mesh

2.3 DiffusionConvectionProblem2D

`DiffusionConvectionReactionProblem2D` is a handle class implementing an abstract diffusion convection reaction problem on a two dimensional polygonal domain. The problem is of the form

$$\begin{cases} -\nabla \cdot (\mu \nabla u) + \beta \cdot \nabla u + \gamma u = f & \text{in } \Omega \\ u = g_D & \text{in } \Gamma_D \\ \mu \frac{\partial}{\partial n} u = g_N & \text{in } \Gamma_N \end{cases} \tag{2.2}$$

where Ω is a two-dimensional open polygonal domain, $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$. The problem can be written in weak form as: find $u_0 \in H^1_{\Gamma_N,0}(\Omega)$ such that

$$a(u_0, v) = F(v) \quad \forall v \in H^1_{\Gamma_N,0}(\Omega) \tag{2.3}$$

where

$$a(u_0, v) := \int_\Omega (\mu \nabla u_0 \cdot \nabla v + \beta \cdot \nabla u_0 v + \gamma u_0 v) d\Omega$$

and

$$F(v) := \int_\Omega f v d\Omega - a(u_g, v) + \int_{\Gamma_N} g_N \gamma_{\Gamma_N}(v) d\Gamma$$

where $u_g \in \{v \in H^1(\Omega) \mid \gamma_{\Gamma_D}(v) = g_D\}$ and the solution u is $u = u_0 + u_g$.

2.3.1 Properties

mu

This property is a scalar value or a function handle containing the function μ in (2.2). If it is a scalar value it means the function μ is constant.

beta

This property is a length-2 array whose components are scalar values or a function handles. It contains the function β in (2.2). If it is a numeric array it means the function β is constant.

gamma

This property is a scalar value or a function handle containing the function γ in (2.2). If it is a scalar value it means the function γ is constant.

f

This property is a function handle containing the function f in (2.2).

domain

This property is a struct containing the description of the domain as requested by the the Barbera's Triangler. It is described in its User's Handbook.

BC

This property is a struct similar to the description of the boundary conditions of the problem requested by the the Barbera's Triangler. The only difference is that `DiffusionConvectionReactionProblem2D.BC` has a new field called `DiffusionConvectionReactionProblem2D.BC.boundaryFunctions` which is a cell array containing at the i-th component the function handle of the function g_D or g_N corresponding to a vertex or a side of the domain with marker i

exactSolution

This property is a struct containing two field:

- `DiffusionConvectionReactionProblem2D.exactSolution.u` is a function handle of the exact solution u of the problem (2.2)
- `DiffusionConvectionReactionProblem2D.exactSolution.uGrad` is a function handle of the gradient ∇u of the exact solution u of the problem (2.2)

2.3.2 Methods

DiffusionConvectionReactionProblem2D()

Constructor of the class `DiffusionConvectionReactionProblem2D`
Parameters:

- `mu`: scalar or function handle
- `beta`: length-2 numeric or function handle array

- `gamma`: scalar or function handle
- `f`: function handle
- `Domain`: struct containing the description of the domain as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `BC`: struct containing the description of the boundary conditions of the problem as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `boundaryFunctions`: cell array containing in the i-th component the function handle of the function g_D or g_N corresponding to a side of the domain with marker i
- `u` (optional): function handle of the exact solution u of the problem (2.2)
- `uGrad` (optional): function handle of the gradient ∇u of the exact solution u of the problem (2.2)

Return:

- `obj`: object of class `DiffusionConvectionReactionProblem2D`

DefineBoundaryConditions()

Method used by the constructor to define the boundary functions onto the vertices of the domain
Parameters:

- `obj`: object of class Element
- `BC`: struct containing the description of the boundary conditions of the problem as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `boundaryFunctions`: cell array containing in the i-th component the function handle of the function g_D or g_N corresponding to a side of the domain with marker i

Return:

- `BC`: struct similar to the description of the boundary conditions of the problem requested by the the Barbera's Triangler. The only difference is that `DiffusionConvectionReactionProblem2D.BC` has a new field called `DiffusionConvectionReactionProblem2D.BC.boundaryFunctions` which is a cell array containing at the i-th component the function handle of the function g_D or g_N corresponding a vertex or a side of the domain with marker i

plotSolution()

Plots the exact solution u of the problem (2.2) on a generic or given mesh
Parameters:

- `obj`: object of class `Element`
- `mesh` (optional): object of class `Mesh`.

Return:

- `fig`: figure object containing the plot of the exact solution of the problem (2.2)

2.4 ApproxDiffusionConvectionProblem2D

`ApproxDiffusionConvectionReactionProblem2D` is a subclass of `DiffusionConvectionReactionProblem2D` implementing the Galerkin Method for diffusion convection reaction problems in a two dimensional polygonal domain. The solution of the Galerkin Discretization is the solution of a problem in variational form: find $u_{0,h} \in \mathbb{V}_h$ such that

$$a(u_{0,h}, v_h) = F(v_h) \quad v_h \in \mathbb{V}_h \tag{2.4}$$

where a and F are defined as in (2.3) and $\mathbb{V}_h \subset H^1_{\Gamma_N,0}(\Omega)$ of finite dimension N_{dof} with a basis given by $\{\varphi_j\}_{1\le j\le N_{\text{dof}}}$. In our case given a conformal triangulation $\mathcal{T}$ we implemented the solution for

$$\mathbb{V}_h = \{v \in H^1_{\Gamma_N,0}(\Omega) \mid v_{|_E} \in \mathbb{P}_i(E) \text{ for all } E \in \mathcal{T}\}$$

with $i = 1, 2$. The approximate solution of (2.2) is $u_h = u_{0,h} + u_g \in V_h$ where $\gamma_{\Gamma_D}(u_g) \simeq g_D$ (in our case an interpolation) and

$$V_h = \{v \in H^1(\Omega) \mid v_{|_E} \in \mathbb{P}_i(E) \text{ for all } E \in \mathcal{T}\}$$

with $i = 1, 2$ and $\dim(V_h) = N_h = N_{\text{dof}} + N_D$ with a basis given by $\{\varphi_j\}_{1\le j\le N_{\text{dof}}} \cup \{\varphi^D_j\}_{1\le j\le N_D}$.

2.4.1 Properties

All the properties of `DiffusionConvectionReactionProblem2D` are properties of `ApproxDiffusionConvectionReactionProblem2D`. Other properties are:

mesh

This property is an object of class `Mesh` describing the triangulation $\mathcal{T}$ where the problem is defined

refElement

This property is a description of a reference element as explained in 2.1.2.

approxSolution

This property is a array of dimension N_h representing a vector $\mathbf{u} = \{u_j\}_{1 \leq j \leq N_h}$ such that the approximate solution u_h can be written as $u_h = \sum_{j=1}^{N_h} u_j \phi_j$ where $\phi_j = \varphi_j$ or $.\phi_j = \varphi_j^D$ depending on if u_j corresponds to a value on a Dirichlet side.

stiffnessMatrix

This property is a matrix $A \in \mathbb{R}^{N_{\text{dof}} \times N_{\text{dof}}}$ with components $A_{j,k} := a(\varphi_k, \varphi_j)$

forcingVector

This property is a vector $\mathbf{f} = \{f_j\}_{1 \leq j \leq N_{\text{dof}}}$ with components $f_j := F(\varphi_j)$

error

This property is a struct with 3 fields

- `L2norm` which is a numerical approximation of $||u - u_h||_{L^2(\Omega)}$.
- `H1seminorm` which is a numerical approximation of $|u - u_h|_{H^1(\Omega)}$.
- `LInfnorm` which is the value of $\max |u_h - h|$ computed on the vertices and middle points of the sides of the triangulation $\mathcal{T}$.

2.4.2 Methods

ApproxDiffusionConvectionReactionProblem2D()

Constructor of the class `ApproxDiffusionConvectionReactionProblem2D`
Parameters:

- `problem`: object of class `DiffusionConvectionReactionProblem2D`
- `mesh`: object of class `Mesh`
- `refElement`: is a description of a reference element as explained in 2.1.2.

Return:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`

generateLinearSystem()

It computes and saves in an object of class `ApproxDiffusionConvectionReactionProblem2D` the properties `ApproxDiffusionConvectionReactionProblem2D.stiffnessMatrix` and `ApproxDiffusionConvectionReactionProblem2D.forcingVector`
Parameters:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`
- `options` (optional): struct with fields
  - `options.massLumping` (boolean). If true it uses mass lumping on the reaction component. Default is true
  - `options.correctConvection` (boolean). If true it uses streamline upwind Petrov-Galerkin method in order to stabilize the numerical solution in the case of high values of convection. Default is true.

solve()

Given the stiffness Matrix A and the vector $\mathbf{f}$, it solves the linear systems $A\mathbf{u}_0 = \mathbf{f}$, it computes the vector $\mathbf{u}$ and saves it in `ApproxDiffusionConvectionReactionProblem2D.approxSolution`.
Parameters:

- `obj`: object of class `Mesh`

getL2Error()

Returns the value stored in `ApproxDiffusionConvectionReactionProblem2D.error.L2norm`. If it is empty, it computes an approximation $||u-u_h||_{L^2(\Omega)}$ using numerical integration and stores it in `ApproxDiffusionConvectionReactionProblem2D.error.L2norm`
Parameters:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`

Return:

- `norm`: numerical approximation of $||u-u_h||_{L^2(\Omega)}$

getH1Error()

Returns the value stored in `ApproxDiffusionConvectionReactionProblem2D.error.H1seminorm`. If it is empty, it computes an approximation $|u-u_h|_{H^1(\Omega)}$ using numerical integration and stores it in `ApproxDiffusionConvectionReactionProblem2D.error.H1seminorm`
Parameters:

- `obj`: object of class ApproxDiffusionConvectionReactionProblem2D

Return:

- `norm`: numerical approximation of $|u-u_h|_{H^1(\Omega)}$.

getLInfError()

Returns the value stored in `ApproxDiffusionConvectionReactionProblem2D.error.LInfnorm`. If it is empty, it computes an approximation $||u-u_h||_{L^\infty(\Omega)}$ as $\max|u_h-h|$ computed on the vertices and middle points of the sides of the triangulation $\mathcal{T}$ and stores it in `ApproxDiffusionConvectionReactionProblem2D.error.LInfnorm`
Parameters:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`

Return:

- `norm`: numerical approximation of $||u-u_h||_{L^\infty(\Omega)}$

plot()

Plots the approximate solution of the problem (2.4)
Parameters:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`

Return:

- `fig`: figure object

2.5 ParabolicProblem

`ParabolicProblem` is a handle class implementing an abstract parabolic problem on a two dimensional polygonal domain. The problem is of the form[1]

$$\begin{cases} \partial_t u - \nabla\cdot(\mu\nabla u) + \beta\cdot\nabla u + \gamma u = f & \text{in } \Omega\times(0,T] \\ u = g_D & \text{in } \Gamma_D\times(0,T] \\ \mu\frac{\partial}{\partial n}u = g_N & \text{in } \Gamma_N\times(0,T] \\ u(0) = u_0 & \text{in } \Omega \end{cases} \tag{2.5}$$

where Ω is a two-dimensional open polygonal domain, $\Gamma_D\cup\Gamma_N = \partial\Omega$ and $\Gamma_D\cap\Gamma_N = \emptyset$. The problem can be written, in the case of homogeneous Dirichlet conditions, in weak form as: find $u\in L^2(0,T,H_0^1(\Omega))\cap C^0([0,T];L^2(\Omega))$ such that

$$\begin{cases} \frac{d}{dt}b(u(t),v) + a(u(t),v) = F(t,v) \quad \forall v\in H_0^1(\Omega),\ \text{q.o. in } t\in(0,T] \\ u(0) = u_0 \end{cases} \tag{2.6}$$

where

$$b(u(t),v) = \int_\Omega u(t)v\,d\Omega;$$

[1] We write $t\in[0,T]$ to make easier the notation but the problem can be defined in any closed interval $[T_b,T_e]$

$$a(u(t), v) := \int_\Omega (\mu \nabla u(t) \cdot \nabla v + \beta \cdot \nabla u(t) v + \gamma u(t) v)\, d\Omega$$

and

$$F(t, v) := \int_\Omega f(t) v\, d\Omega$$

where $f \in L^2(0, T; L^2(\Omega))$ and $u_0 \in L^2(\Omega)$

2.5.1 Properties

mu

This property is a scalar value or a function handle containing the function μ in (2.5). If it is a scalar value it means the function μ is constant. μ does not depends on t

beta

This property is a length-2 array whose components are scalar values or a function handles. It contains the function β in (2.5). If it is a numeric array it means the function β is constant. β does not depends on t

gamma

This property is a scalar value or a function handle containing the function γ in (2.2). If it is a scalar value it means the function γ is constant. γ does not depends on t

f

This property is a function handle containing the function f in (2.5).

domain

This property is a struct containing the description of the domain as requested by the the Barbera's Triangler. It is described in its User's Handbook.

timeInterval

This property is a length-2 array whose components are the starting and stopping time of the time interval where the problem is defined

BC

This property is a struct similar to the description of the boundary conditions of the problem requested by the the Barbera's Triangler. The only difference is that `ParabolicProblem.BC` has a new field called `ParabolicProblem.BC.boundaryFunctions` which is a cell array containing at the i-th component the function handle of the function g_D or g_N corresponding a vertex or a side of the domain with marker i

exactSolution

This property is a struct containing two field:

- `DiffusionConvectionReactionProblem2D.exactSolution.u` is a function handle of the exact solution u of the problem (2.5)
- `DiffusionConvectionReactionProblem2D.exactSolution.uGrad` is a function handle of the gradient ∇u of the exact solution u of the problem (2.5)

2.5.2 Methods

ParabolicProblem()

Constructor of the class `ParabolicProblem`
Parameters:

- `mu`: scalar or function handle
- `beta`: length-2 numeric or function handle array
- `gamma`: scalar or function handle
- `f`: function handle
- `domain`: struct containing the description of the domain as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `timeInterval`: length-2 array whose components are the starting and stopping time of the time interval where the problem is defined
- `BC`: struct containing the description of the boundary conditions of the problem as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `boundaryFunctions`: cell array containing in the i-th component the function handle of the function g_D or g_N corresponding to a side of the domain with marker i
- `u` (optional): function handle of the exact solution u of the problem (2.2)
- `uGrad` (optional): function handle of the gradient ∇u of the exact solution u of the problem (2.2)

Return:

- `obj`: object of class `ParabolicProblem`

DefineBoundaryConditions()

Method used by the constructor to define the boundary functions onto the vertices of the domain
Parameters:

- `obj`: object of class `Element`
- `BC`: struct containing the description of the boundary conditions of the problem as requested by the the Barbera's Triangler. It is described in its User's Handbook.
- `boundaryFunctions`: cell array containing in the i-th component the function handle of the function g_D or g_N corresponding to a side of the domain with marker i

Return:

- `BC`: struct similar to the description of the boundary conditions of the problem requested by the the Barbera's Triangler. The only difference is that `ParabolicProblem.BC` has a new field called `ParabolicProblem.BC.boundaryFunctions` which is a cell array containing at the i-th component the function handle of the function g_D or g_N corresponding a vertex or a side of the domain with marker i

2.6 ApproxParabolicProblem

`ApproxParabolicProblem` is a subclass of `ParabolicProblem` implementing the method of lines for parabolic problems in a two dimensional polygonal domain discretized using the Galerkin Method. The solution of the Galerkin Discretization is the solution of a problem, for simplicity in the case of homogeneus dirichlet conditions[2], in variational form: find $u_h \in C^0([0,T];\mathbb{V}_h)$ such that

$$\begin{cases} \frac{d}{dt}b(u_h(t),v_h) + a(u_h(t),v_h) = F(t,v_h) & \forall v \in \mathbb{V}_h, \text{ q.o. in } (0,T] \\ u(0) = u_{0,h} & u_{0,h} \in \mathbb{V}_h \end{cases} \tag{2.7}$$

where b, a and F are defined as in (2.6), $\mathbb{V}_h \subset H_0^1(\Omega)$ of finite dimension N_{dof} with a basis given by $\{\varphi_j\}_{1\le j\le N_{\text{dof}}}$ and $u_{0,h}$ is an approximation of u_0 (in our case it is the interpolation). Given a conformal triangulation $\mathcal{T}$ we implemented the solution for

$$\mathbb{V}_h = \{v \in H_0^1(\Omega) \mid v_{|_E} \in \mathbb{P}_i(E) \text{ for all } E \in \mathcal{T}\}$$

with $i = 1, 2$. The ODE obtained is then solved numerically using Crank–Nicolson method.

[2] the code deals also with non-homogenous Dirichlet and Neumann conditions

2.6.1 Properties

All the properties of `ParabolicProblem` are properties of`ApproxParabolicProblem`. Other properties are:

mesh

This property is an object of class `Mesh` describing the triangulation $\mathcal{T}$ where the problem is defined

timeDiscretization

This property is an array $\{t_0, t_1, \ldots, t_{N_t}\}$ of length $N_t + 1$ with the time values of the discretization of the time interval

refElement

This property is a description of a reference element as explained in 2.1.2.

approxSolution

This property is an array of dimension $N_h \times N_t$ representing the vectors $\mathbf{u_n} = \{u_{j,n}\}$ such that the approximate solution u_h at time t_n can be written as $u_h^n = \sum_{j=1}^{N_h} u_{j,n}\phi_j$ where $\phi_j = \varphi_j$ or $\phi_j = \varphi_j^D$ depending on if $u_{j,n}$ corresponds to a value on a Dirichlet side.

stiffnessMatrix

This property is a matrix $A \in \mathbb{R}^{N_{\text{dof}} \times N_{\text{dof}}}$ whose components $A_{j,k} := a(\varphi_k, \varphi_j)$

Ad

This property is a matrix $A^D \in \mathbb{R}^{N_{dof} \times N_D}$ whose components $A^D_{j,k_D} := a(\varphi^D_{k_D}, \varphi_j)$.

massMatrix

This property is a matrix $B \in \mathbb{R}^{N_{\text{dof}} \times N_{\text{dof}}}$ whose components $B_{j,k} := b(\varphi_k, \varphi_j)$

Md

This property is a matrix $B^D \in \mathbb{R}^{N_{dof} \times N_D}$ whose components $B^D_{j,k_D} := b(\varphi^D_{k_D}, \varphi_j)$.

forcingVector

This property is a matrix where each column is vector $\mathbf{f^n} = \{f_j^n\}$ whose components $f_j^n := F(t_n, \varphi_j)$

Gn

This property is a matrix where each column is vector $\mathbf{g^n} = \{g_j^n\}$ whose components $g_j^n := \int_{\Gamma_N} g_N(t_n)\varphi_j \, d\Gamma$

error

This property is a struct with 3 fields

- `L2norm` which is a numerical approximation of $||u(T) - u_h(T)||_{L^2(\Omega)}$.
- `H1seminorm` which is a numerical approximation of $|u(T) - u_h(T)|_{H^1(\Omega)}$.
- `LInfnorm` which is the value of $\max |u_h(T) - h(T)|$ computed on the vertices and middle points of the sides of the triangulation $\mathcal{T}$.

2.6.2 Methods

ApproxParabolicProblem()

Constructor of the class `ApproxParabolicProblem`
Parameters:

- `problem`: object of class `ParabolicProblem`
- `mesh`: object of class `Mesh`
- `refElement`: is a description of a reference element as explained in 2.1.2.
- `timeDiscretization`: number of time steps for a uniform discretization of the time intervals or array of the time values of the discretization.

Return:

- `obj`: object of class `ApproxParabolicProblem`

computeMassMatrix()

It computes and saves in an object of class `ApproxParabolicProblem` the properties `ApproxParabolicProblem.massMatrix` and `ApproxParabolicProblem.Bd`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`
- `options` (optional): struct with fields
  - `options.massLumping` (boolean). If true it uses mass lumping to make the matrix diagonal. Default is false

computeStiffnessMatrix()

It computes and saves in an object of class `ApproxParabolicProblem` the properties `ApproxParabolicProblem.stiffnessMatrix` and `ApproxParabolicProblem.Ad`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`
- `options` (optional): struct with fields
  - `options.massLumping` (boolean). If true it uses mass lumping for the reaction component. Default is false
  - `options.correctConvection` (boolean). If true it uses streamline upwind Petrov-Galerkin method for correcting high values of convection. Default is true.

computeF()

It computes and saves a new column in the property `ApproxParabolicProblem.forcingVector`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`
- `t`: time value in which to compute the vector
- `options` (optional): struct with fields
  - `options.correctConvection` (boolean). If true it uses streamline upwind Petrov-Galerkin method for correcting high values of convection. Default is true.

computeGn()

It computes and saves a new column in the property `ApproxParabolicProblem.Gn`
`ApproxDiffusionConvectionReactionProblem2D.forcingVector`
Parameters:

- `obj`: object of class `ApproxDiffusionConvectionReactionProblem2D`
- `t`: time value in which to compute the vector

solve()

Given the matrices A, A^D, B and B^D, it solves numerically the ODE (2.7) using the Crank-Nicolson method. It saves the results in `ApproxParabolicProblem.approxSolution`.
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

getL2Error()

Returns the value stored in `ApproxParabolicProblem.error.L2norm`. If it is empty, it computes an approximation $||u(T) - u_h(T)||_{L^2(\Omega)}$ using numerical integration and stores it in `ApproxParabolicProblem.error.L2norm`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

Return:

- `norm`: numerical approximation of $||u(T) - u_h(T)||_{L^2(\Omega)}$

getH1Error()

Returns the value stored in `ApproxParabolicProblem.error.H1seminorm`. If it is empty, it computes an approximation $|u(T) - u_h(T)|_{H^1(\Omega)}$ using numerical integration and stores it in `ApproxParabolicProblem.error.H1seminorm`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

Return:

- `norm`: numerical approximation of $|u(T) - u_h(T)|_{H^1(\Omega)}$.

getLInfError()

Returns the value stored in `ApproxParabolicProblem.error.LInfnorm`. If it is empty, it computes an approximation $||u(T) - u_h(T)||_{L^\infty(\Omega)}$ as $\max |u_h(T) - u(T)|$ computed on the vertices and middle points of the sides of the triangulation $\mathcal{T}$ and stores it in `ApproxParabolicProblem.error.LInfnorm`
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

Return:

- `norm`: numerical approximation of $||u(T) - u_h(T)||_{L^\infty(\Omega)}$

plot()

Plots the approximate solution of the problem (2.7) at time T
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

Return:

- `fig`: figure object

video()

Makes an animation the approximate solution of the problem (2.7)
Parameters:

- `obj`: object of class `ApproxParabolicProblem`

Return:

- `M`: movie object

Chapter 3

Tests

In this chapter we will see the results of our approximations on some problems and we will test the order of convergence.

3.1 Diffusion Convection Reaction problems with Dirichlet conditions

In this section we will test problems of the form:

$$\begin{cases} \nabla \cdot (\mu \nabla u) + \beta \cdot \nabla u + \gamma u = f & \text{in } \Omega \\ u = g_D & \text{in } \partial\Omega \end{cases}$$

3.1.1 Problem1

The first problem is a diffusion problem with homogeneous Dirichlet conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = 0 & \text{in } \partial\Omega \end{cases} \tag{3.1}$$

with

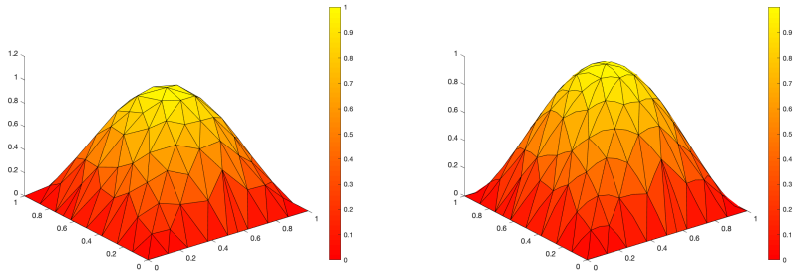
$$f := 32(x(1-x) + y(1-y))$$

which has exact solution

$$u = 16x(1-x)y(1-y).$$

Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	2.1220	0.9966	-0.9316
	H^1	1.0627	0.4991	-0.4666
P2	L^2	3.2314	1.5196	-1.4760
	H^1	2.1655	1.0185	-0.9892



(a) *P1* (b) *P2*

Figure 3.1: Numerical approximation of the solution u using P1 and P2 elements

3.1.2 Problem2

The second problem is a diffusion problem with non-homogeneous Dirichlet conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = xy & \text{in } \partial\Omega \end{cases}$$

with

$$f := 32(x(1-x) + y(1-y))$$

which has exact solution

$$u = 16x(1-x)y(1-y) + xy$$

Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	2.1158	0.9938	-0.9289
	H^1	1.0581	0.4970	-0.4646
P2	L^2	3.2314	1.5196	-1.4760
	H^1	2.1655	1.0185	-0.9892

3.1.3 Problem3

The third problem is a diffusion problem with non-homogeneous Dirichlet conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = \sin(2\pi xy) & \text{in } \partial\Omega \end{cases}$$

with

$$f := 4\pi^2(x^2 + y^2)\sin(2\pi xy)$$

which has exact solution

$$u = \sin(2\pi xy)$$

Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	2.0896	0.9864	-0.9212
	H^1	1.0454	0.4929	-0.4600
P2	L^2	3.0904	1.4539	-1.4111
	H^1	2.0412	0.9601	-0.9319

3.1.4 Problem4

The fourth problem is a diffusion convection problem with non-homogeneous Dirichlet conditions:

$$\begin{cases} -\mu\Delta u + \beta \cdot \nabla u = f & \text{in } \Omega \\ u = \sin(2\pi xy) & \text{in } \partial\Omega \end{cases}$$

with $\mu = 0.0001$, $\beta = 1000000(1,1)^T$ and

$$f := 0.0004\pi^2(x^2+y^2)\sin(2\pi xy) + 2000000\pi(x+y)\cos(2\pi xy)$$

which has exact solution

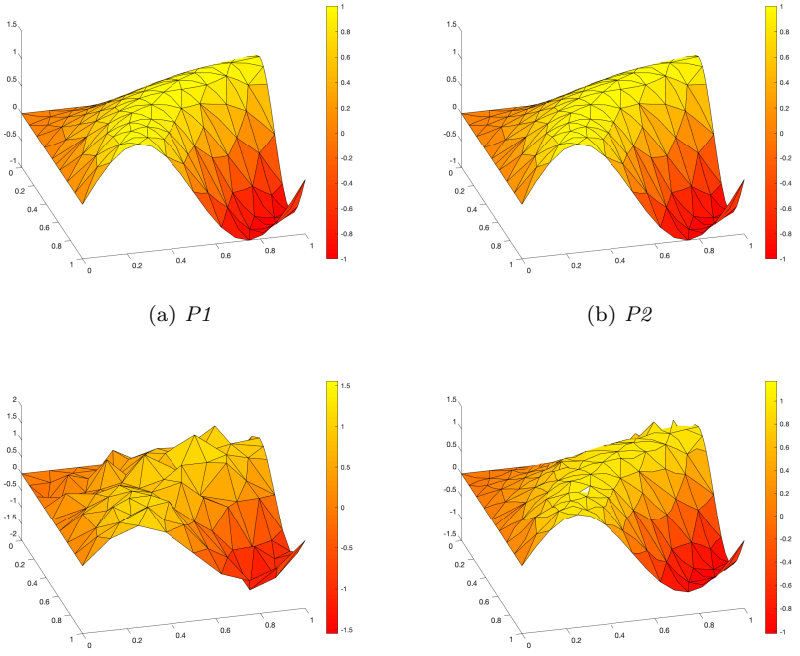
$$u = \sin(2\pi xy)$$

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	2.1996	1.0366	-0.9679
	H^1	1.0365	0.4886	-0.4560
P2	L^2	3.0732	1.4450	-1.4027
	H^1	2.0275	0.9536	-0.9257

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	-3.7446	-1.6350	1.5317
	H^1	-4.8274	-2.1457	2.0086
P2	L^2	1.1715	0.3995	-0.3825
	H^1	0.1020	-0.1042	0.1062



(a) *P1*

(b) *P2*

(c) *P1 without correction*

(d) *P2 without correction*

Figure 3.2: Numerical approximation of the solution u using P1 and P2 elements with and without Upwind Petrov-Galerkin correction

3.1.5 Problem5

The fifth problem is a diffusion reaction problem with non-homogeneous Dirichlet conditions:

$$\begin{cases} -\mu\Delta u + \sigma u = f & \text{in } \Omega \\ u = \sin(2\pi xy) & \text{in } \partial\Omega \end{cases}$$

with $\mu = 0.0001$, $\sigma = 1000000$ and

$$f := 0.0004\pi^2(x^2 + y^2)\sin(2\pi xy) + 1000000\sin(2\pi xy)$$

which has exact solution

$$u = \sin(2\pi xy)$$

Numerical results on the orders of convergence with P1 and P2 elements using the mass lumping correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	1.5182	0.7150	-0.6680
	H^1	0.3446	0.1615	-0.1515
P2	L^2	3.2637	1.5369	-1.4916
	H^1	2.2559	1.0623	-1.0309

Numerical results on the orders of convergence with P1 and P2 elements without using the mass lumping correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	2.2514	1.0615	-0.9910
	H^1	1.0741	0.5066	-0.4728
P2	L^2	3.0559	1.4378	-1.3955
	H^1	2.0509	0.9647	-0.9364

3.1.6 Problem6

The sixth problem is a diffusion convection problem with non-homogeneous Dirichlet conditions:

$$\begin{cases} -\mu\Delta u + \beta \cdot \nabla u = f & \text{in } \Omega \\ u = \sin(2\pi xy) & \text{in } \partial\Omega \end{cases}$$

with $\mu = 0.0001$, $\beta = 1000000(sin(xy), e^y)^T$ and

$$f := 0.0004\pi^2(x^2 + y^2)\sin(2\pi xy) + 2000000\pi(\sin(xy)y + e^y x)\cos(2\pi xy)$$

which has exact solution

$$u = \sin(2\pi xy)$$

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	2.2713	1.0689	-0.9993
	H^1	1.0595	0.4994	-0.4663
P2	L^2	3.0480	1.4319	-1.3901
	H^1	2.0005	0.9405	-0.9130

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	1.7551	0.8251	-0.7733
	H^1	0.6890	0.3214	-0.3029
P2	L^2	2.1998	1.0342	-1.0051
	H^1	1.2140	0.5713	-0.5554

3.2 Diffusion Convection Reaction problems with Dirichlet and Neumann conditions

In this section we will test problems of the form:

$$\begin{cases} \nabla \cdot (\mu \nabla u) + \beta \cdot \nabla u + \gamma u = f & \text{in } \Omega \\ u = g_D & \text{in } \Gamma_D \\ \mu \frac{\partial}{\partial n} u = g_N & \text{in } \Gamma_N \end{cases}$$

where

$$\Omega = (0,1) \times (0,1),$$
$$\Gamma_D = \{0\} \times [0,1] \cup [0,1] \times \{0\}$$

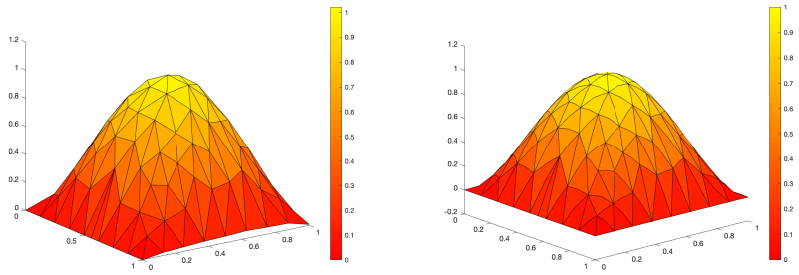
and

$$\Gamma_N = \{1\} \times (0,1] \cup (0,1] \times \{1\}.$$

3.2.1 ProblemNeumann1

The first problem is a diffusion problem with homogeneous Dirichlet conditions and non homogeneous Neumann Conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = 0 & \text{in } \Gamma_D \\ \frac{du}{dn} = -16y(1-y) & \text{in} \{1\} \times (0,1) \\ \frac{du}{dn} = -16x(1-x) & \text{in} (0,1) \times \{1\} \end{cases}$$



(a) *P1* (b) *P2*

Figure 3.3: Numerical approximation of the solution u using P1 and P2 elements

with

$$f := 32(x(1-x) + y(1-y))$$

which has exact solution

$$u = 16x(1-x)y(1-y).$$

Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	*Area*	N_{dof}
P1	L^2	2.0323	0.9671	-0.9789
	H^1	1.0567	0.4964	-0.5021
P2	L^2	3.2110	1.5096	-1.5225
	H^1	2.1530	1.0125	-1.0211

3.2.2 ProblemNeumann2

The second problem is a diffusion convection problem with non-homogeneous Dirichlet and Neumann conditions:

$$\begin{cases} -\mu\Delta u + \beta \cdot \nabla u = f & \text{in } \Omega \\ u = \sin(2\pi xy) & \text{in } \Gamma_D \\ \mu\frac{du}{dn} = 2\pi y\cos(2\pi y) & \text{in } \{1\} \times (0,1) \\ \mu\frac{du}{dn} = 2\pi x\cos(2\pi x) & \text{in } (0,1) \times \{1\} \end{cases}$$

with $\mu = 0.0001$, $\beta = 1000000(1,1)^T$ and

$$f := 0.0004\pi^2(x^2+y^2)\sin(2\pi xy) + 2000000\pi(x+y)\cos(2\pi xy)$$

which has exact solution

$$u = \sin(2\pi xy)$$

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	2.1647	1.0191	-1.0306
	H^1	1.0388	0.4897	-0.4957
P2	L^2	3.1184	1.4657	-1.4785
	H^1	2.0867	0.9815	-0.9901

Numerical results on the orders of convergence with P1 and P2 elements using the upwind Petrov-Galerkin correction can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	1.2430	0.5794	-0.5855
	H^1	0.3191	0.1447	-0.1453
P2	L^2	1.9966	0.9441	-0.9533
	H^1	0.9753	0.4649	-0.4700

3.3 Parabolic problem with Dirichlet boundary conditions

In this section we will test problems of the form:

$$\begin{cases} \partial_t u + \nabla \cdot (\mu \nabla u) = f & \text{in } \Omega \times (0,T] \\ u = g_D & \text{in } \partial\Omega \times (0,T] \\ u(0) = u_0 & \text{in } \Omega \times \{0\} \end{cases}$$

3.3.1 ProblemT

The first problem is a heat problem with homogeneous Dirichlet conditions:

$$\begin{cases} \partial_t u - \Delta u = f(t) & \text{in } \Omega \times (0,T] \\ u = x\sin(10t) + y\cos(5t) & \text{in } \partial\Omega \times (0,T] \\ u(0) = y & \text{in } \Omega \end{cases} \tag{3.2}$$

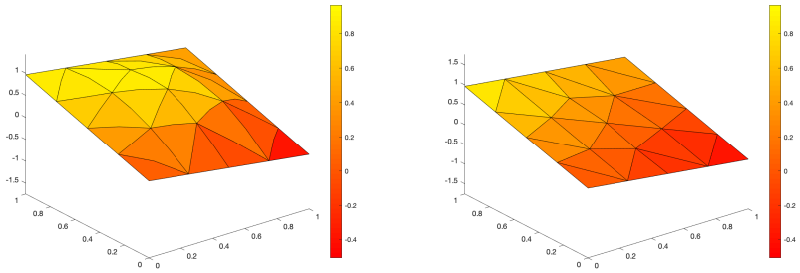
with

$$f(t) := 10x\cos(10t) - 5y\sin(5t)$$

which has exact solution

$$u = x\sin(10t) + y\cos(5t)$$

We notice that that the exact solution u of this problem is always a plane for each t, but it depends complexly from t, therefore it is good for testing the



(a) *Coarse time discretization* (b) *Fine time discretization*

Figure 3.4: Numerical approximation of the solution $u(T)$ with a coarse and a finer temporal discretization

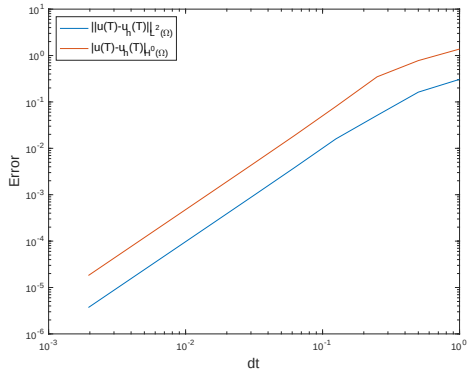


Figure 3.5: LogLog plot of the L^2 norm and H^1 seminorm of the error varying the value of Δt

convergence in terms of the time discretization. Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		Δt
P1	L^2	1.8697
	H^1	1.8740
P2	L^2	1.8785
	H^1	1.8850

3.3.2 ProblemX

The second problem is a heat problem with homogeneous Dirichlet conditions:

$$\begin{cases} \partial_t u - \Delta u = f(t) & \text{in } \Omega \times (0,T] \\ u = t(e^x - xy^2) & \text{in } \partial\Omega \times (0,T] \\ u(0) = 0 & \text{in } \Omega \end{cases} \tag{3.3}$$

with

$$f(t) := e^x - xy^2 - t * (e^x - 2x)$$

which has exact solution

$$u = t(e^x - xy^2)$$

We notice that that the exact solution u of this problem depends linearly from t therefore it is good for testing the convergence in terms of the space discretization. Numerical results on the orders of convergence with P1 and P2 elements can be seen in the following table: L^2 refers to the $L^2(\Omega)$ norm of the error, while H^1 refers to the $H^1(\Omega)$ seminorm of the error.

		h	$Area$	N_{dof}
P1	L^2	2.1359	1.0032	-0.9388
	H^1	1.0516	0.4949	-0.4625
P2	L^2	3.1876	1.4997	-1.4562
	H^1	2.1217	0.9980	-0.9691

3.4 Comments on the numerical results

The expected rate of convergence in the Finite Element Method for an elliptic problem using a $\mathbb{P}_k$ element is

$$||u - u_h||_{L^2(\Omega)} \sim Ch^{k+1},$$
$$|u - u_h|_{H^0(\Omega)} \sim Ch^k$$

where h is the biggest side of the triangulation $\mathcal{T}$. If the triangulation $\mathcal{T}$ is regular we have Area $\sim h^2$ therefore we can write

$$||u - u_h||_{L^2(\Omega)} \sim C\text{Area}^{\frac{k+1}{2}},$$
$$|u - u_h|_{H^0(\Omega)} \sim C\text{Area}^{\frac{k}{2}}.$$

The number of degrees of freedom N_{dof} scales as h^{-2}, therefore we obtain

$$||u - u_h||_{L^2(\Omega)} \sim CN_{\text{dof}}^{-\frac{k+1}{2}},$$
$$|u - u_h|_{H^0(\Omega)} \sim CN_{\text{dof}}^{-\frac{k}{2}}.$$

Comparing the theoretical rate of convergence with the numerical estimate using the `polyfit` function in Matlab (which are summarized in the tables above), we notice that the results are nearly always compatible. There are, anyway, some important but sometimes expected, exceptions:

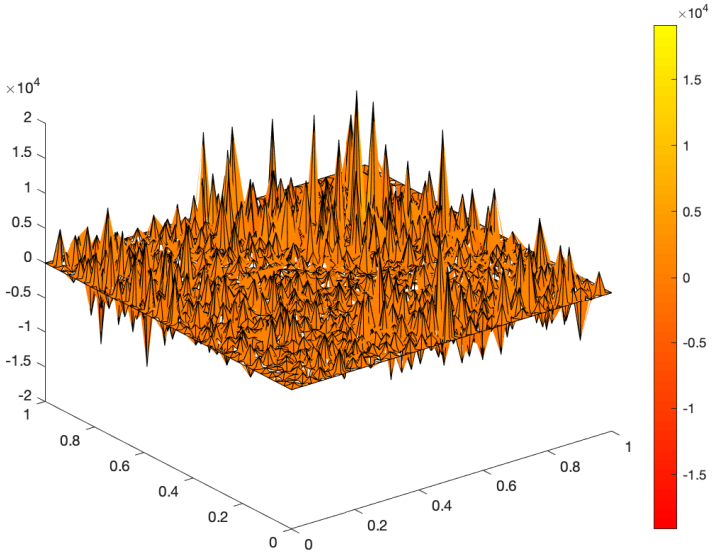


Figure 3.6: Numerical approximation of the solution u for problem 5 using P2 elements and msss lumping

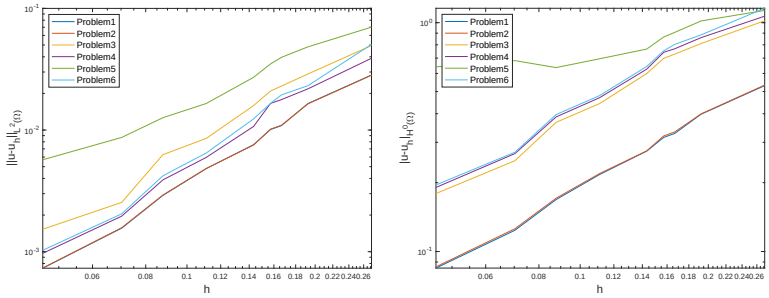
- In problem 4, given the high convection and small diffusion, the method without any corrections fail to converge when using $\mathbb{P}_1$ elements and converge much slower than expected in the $\mathbb{P}_2$ case.

- In problem 5, when using mass lumping the rate of convergence is slower than expected with the $\mathbb{P}_1$ elements and, even if the rate of convergence seems to be the expected one, the numerical solution fails to converge with the $\mathbb{P}_2$ elements as can be seen in figure 3.6. The fact that with $\mathbb{P}_2$ elements fail converge using mass lumping is not surprising given the fact that mass lumping is a correction of the first order, anyway it is surprising that without using mass lumping the method converges.

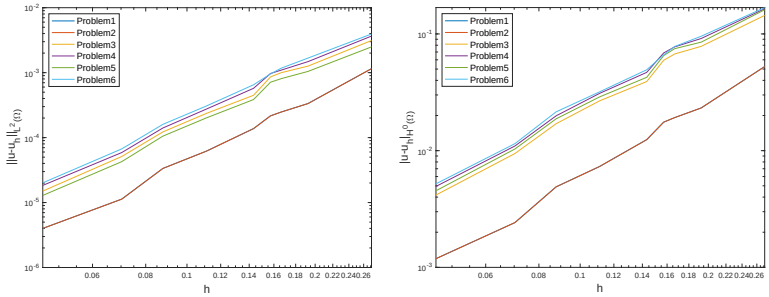
- In problem 6 and problem neumann 2, both with $\mathbb{P}_1$ and $\mathbb{P}_2$ elements when it is not used the upwind Petrov-Galerkin correction the method converge at a slower pace.

The expected rate of convergence in the method of lines for an parabolic problem using a $\mathbb{P}_k$ element for the spacial discretization and Crank-Nicolson method for solving numerically the resulting ODE is

$$||u(T) - u_h^{N_t}||_{L^2(\Omega)} \sim C_1 \Delta t^2 + C_2 h^{k+1},$$
$$|u(T) - u_h^{N_t}|_{H^0(\Omega)} \sim C_1 \Delta t^2 + C_2 h^k.$$

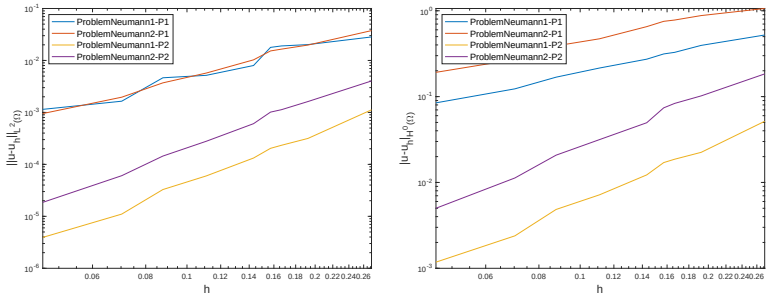


(a) *P1*-L^2 (b) *P1*-H^0



(c) *P2*-L^2 (d) *P2*-H^0

Figure 3.7: LogLog plot of the norms of the errors on the elliptic problems with Dirichlet boundary conditions. For problem 4 and 6 we plotted the results using the upwind Petrov-Galerkin correction and for problem 5 we plotted the results using mass lumping in the case of $\mathbb{P}_1$ elements and without using mass lumping in the case of $\mathbb{P}_2$ elements. We notice that, excluding problem 5 in the $\mathbb{P}_1$ case which has a slower rate of convergence, the lines are all approximately parallel



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

Figure 3.8: LogLog plot of the norms of the errors on the elliptic problems with Dirichlet and Neumann boundary conditions. For problem neumann 2 we plotted the results using the upwind Petrov-Galerkin correction. The lines corresponding to the same kind of elements are all approximately parallel

We have tested the rate of convergence on the temporal discretization in ProblemT and the spacial rate of convergence in ProblemX and the results are compatible with the theoretical results