Notes ⋆ Analyse Numérique - Part II: Implementation of the Finite Elements ⋆ Examination - Numerical Validation

Nguyen Quan Ba Hong*

November 9, 2018

Abstract

This context includes the materials given in the course *Finite Element Method* in the Master 2 Fundamental Mathematics program 2018-2019, with my MATLAB scripts provided.

Brief introduction. "This lecture is a numerical counterpart to *Sobolev spaces & elliptic equations*. In the first part, after some reminders on linear elliptic partial differential equations, the approximation of the associated solutions by the finite element methods is investigated. Their construction and their analysis are described in one and two dimensions. The second part of the lectures consists in defining a generic strategy for the implementation of the method based on the variational formulation. A program is written in MATLAB (implementable with MATLAB or OCTAVE)."

*Master 2 Student at UFR mathématiques, Université de Rennes 1, Beaulieu - Bâtiment 22 et 23, 263 avenue du Général Leclerc, 35042 Rennes CEDEX, France.
E-mail: nguyenquanbahong@gmail.com
Blog: www.nguyenquanbahong.com

Contents

1 Validation of the Assembling Procedure **3**

2 Validation of the Convergence Order of the Finite-Element Method **7**

3 Appendices: MATLAB Scripts **15**

3.1 Function One 15

3.2 Exact Solution 16

3.3 Source Function 16

3.4 Solve the Neumann Boundary Value Problem 16

3.5 Convergence Order 18

Introduction

To validate a code, several strategies are used:

1. Check each function or elementary part of the code by application to very simple examples where the results can be verified by hand calculation.
2. Check the obtained results in configurations where the exact solution is known.
3. Check the asymptotic behavior. For example, you can check the asymptotic behavior of the finite-element error with respect to the discretization parameter h. In the case of a PDE the solution of which is known, you will plot the error with respect to the discretization parameter h in "loglog" scale.

The following exercises which involve these strategies are proposed.

1 Validation of the Assembling Procedure

In order to validate your programming in the context of iso-parametric $\mathbb{P}_1$ finite elements, you will write a script which calculates the mass and stiffness matrices for the following mesh:

```
coords_ = [0,0;.5,0;1,0;0,.5;.5,.5;1,.5;0,1;.5,1;1,1];
triangles_ = [1,2,4;2,5,4;2,3,5;3,6,5;4,5,7;5,8,7;5,6,8;6,9,8];
```

To be sure of the correctness of the numerical results, you can calculate some components of the matrices by hand. The results and your analysis of them will be written in the script as ending comments. Finally, you will also justify the choice of the quadrature rule for each matrix. This justification will be expressed in the script, as a comment where it appears to be the most relevant for you.

N.B.: We remember the definition of the mass matrix

$$M_{ij} = \int_{\Omega_h} \varphi_i(x)\, \varphi_j(x)\, dx, \quad \forall (i,j), \tag{1.1}$$

and the stiffness matrix

$$R_{ij} = \int_{\Omega_h} \nabla\varphi_i(x) \cdot \nabla\varphi_j(x)\, dx, \quad \forall (i,j), \tag{1.2}$$

where (i,j) designates the couples of nodes of the mesh, Ω_h the domain as described by the triangles of the mesh and $\{\varphi_i\}_i$ the basis functions of the finite-element interpolation.

Running the following codes in the main script

```
%% Compute the Mass Matrix
% Use the P_1 nodal quadrature formula
```

```
M =  assemble_matrix('Id', 'Id', 'f_one', mesh_geo, 1, 1)
% Use the P_3 nodal quadrature formula
M =  assemble_matrix('Id', 'Id', 'f_one', mesh_geo, 1, 2)

%% Compute the Stiffness Matrix
R1 = assemble_matrix('D1', 'D1', 'f_one', mesh_geo, 1, 1);
R2 = assemble_matrix('D2', 'D2', 'f_one', mesh_geo, 1, 1);
R = R1 + R2
```

gives us the following mass and stiffness matrices:

```
M =

  Columns 1 through 3

   0.041666666666667                   0                   0
                   0   0.125000000000000                   0
                   0                   0   0.083333333333333
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0

  Columns 4 through 6

                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
   0.125000000000000                   0                   0
                   0   0.250000000000000                   0
                   0                   0   0.125000000000000
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0

  Columns 7 through 9

                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
```

```
                  0                   0                   0
                  0                   0                   0
                  0                   0                   0
   0.083333333333333                  0                   0
                  0   0.125000000000000                   0
                  0                   0   0.041666666666667
```

```
M =

  Columns 1 through 3

   0.020833333333333   0.010416666666667                   0
   0.010416666666667   0.062500000000000   0.010416666666667
                  0   0.010416666666667   0.041666666666667
   0.010416666666667   0.020833333333333                   0
                  0   0.020833333333333   0.020833333333333
                  0                   0   0.010416666666667
                  0                   0                   0
                  0                   0                   0
                  0                   0                   0

  Columns 4 through 6

   0.010416666666667                   0                   0
   0.020833333333333   0.020833333333333                   0
                  0   0.020833333333333   0.010416666666667
   0.062500000000000   0.020833333333333                   0
   0.020833333333333   0.125000000000000   0.020833333333333
                  0   0.020833333333333   0.062500000000000
   0.010416666666667   0.020833333333333                   0
                  0   0.020833333333333   0.020833333333333
                  0                   0   0.010416666666667

  Columns 7 through 9

                  0                   0                   0
                  0                   0                   0
                  0                   0                   0
   0.010416666666667                  0                   0
   0.020833333333333   0.020833333333333                   0
                  0   0.020833333333333   0.010416666666667
```

```
   0.041666666666667   0.010416666666667                   0
   0.010416666666667   0.062500000000000   0.010416666666667
                   0   0.010416666666667   0.020833333333333


R =

  Columns 1 through 3

   1.000000000000000  -0.500000000000000                   0
  -0.500000000000000   2.000000000000000  -0.500000000000000
                   0  -0.500000000000000   1.000000000000000
  -0.500000000000000                   0                   0
                   0  -1.000000000000000                   0
                   0                   0  -0.500000000000000
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0

  Columns 4 through 6

  -0.500000000000000                   0                   0
                   0  -1.000000000000000                   0
                   0                   0  -0.500000000000000
   2.000000000000000  -1.000000000000000                   0
  -1.000000000000000   4.000000000000000  -1.000000000000000
                   0  -1.000000000000000   2.000000000000000
  -0.500000000000000                   0                   0
                   0  -1.000000000000000                   0
                   0                   0  -0.500000000000000

  Columns 7 through 9

                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
  -0.500000000000000                   0                   0
                   0  -1.000000000000000                   0
                   0                   0  -0.500000000000000
   1.000000000000000  -0.500000000000000                   0
  -0.500000000000000   2.000000000000000  -0.500000000000000
                   0  -0.500000000000000   1.000000000000000
```

which are exactly the results given in the script `test_assemble_matrix_etu_0.m` in [**1**]. □

Remark 1.1. *To compute the mass matrix M, the $\mathbb{P}_3$ nodal quadrature formula must be used rather than the $\mathbb{P}_1$ nodal quadrature formula, since the polynomials in the involving integrand has the degree two.*

Moreover, the Gaussian quadrature formulas can be used instead of the nodal quadrature formulas to increase the accuracy of the assembling procedure for the finite-element right hand side vector.

2 Validation of the Convergence Order of the Finite-Element Method

We consider the problem

$$\begin{cases} -\Delta u + u = f, & \text{on } \Omega, \\ \dfrac{\partial u}{\partial n} = 0, & \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

where Ω is the square $[0,1] \times [0,1]$, and the function f is chosen such that the exact solution is

$$u(x) = \cos(\pi x_1) \cos(\pi x_2). \tag{2.2}$$

Problem 2.1. *Write a function which solves the problem by application of the finite-element method (iso-parametric $\mathbb{P}_1$ or $\mathbb{P}_2$) with the following entries: the discretization parameter h of the mesh and the degree of the interpolation. This function builds or loads the mesh ($\mathbb{P}_1$ or $\mathbb{P}_2$) according to the discretization parameter h, then assemblies the matrix of the variational approximation, and finally solves the linear system. The output of the function is a vector which contains the approximation of u at the nodes of the mesh, and the coordinates of these nodes.*

Solution. See Sec. 3.4. Running the following command lines in the main script

```
[X, nodes] = solve_Neumann_BVP(mesh_geo, 100, degree_FE);
figure
trisurf(mesh_geo.triangles, mesh_geo.coords(:,1), mesh_geo.coords(:,2), X);
```

with `degree_FE=1,2` and some refinements, gives us

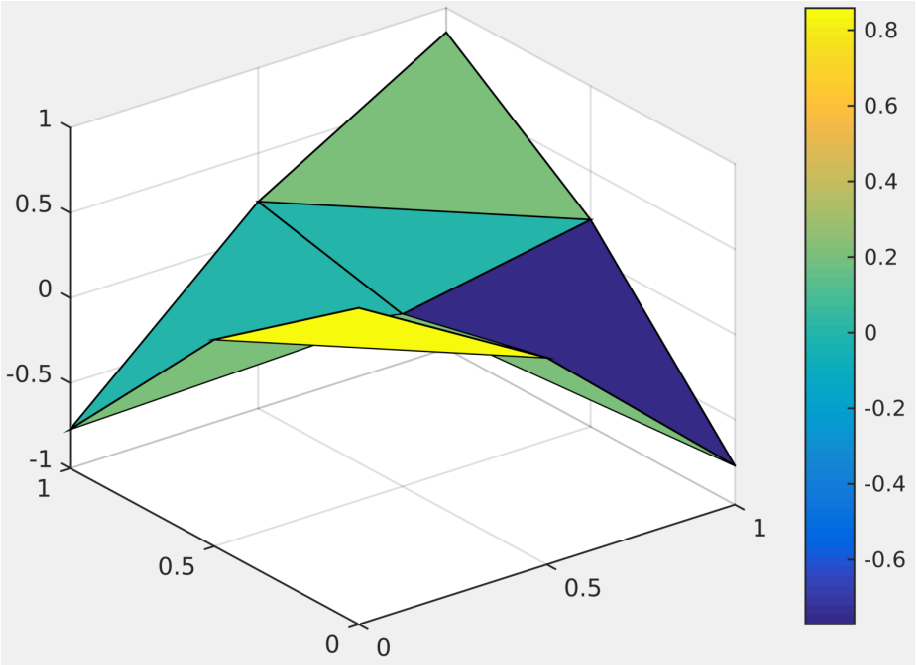

Figure 1: $\mathbb{P}_1$ approximate solution.

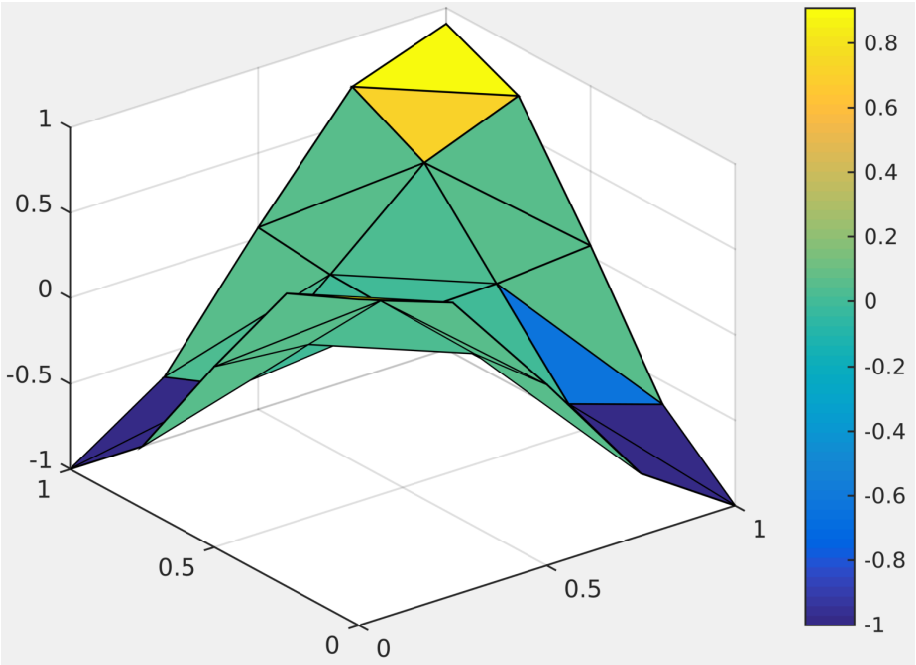

Figure 2: $\mathbb{P}_1$ approximate solution.

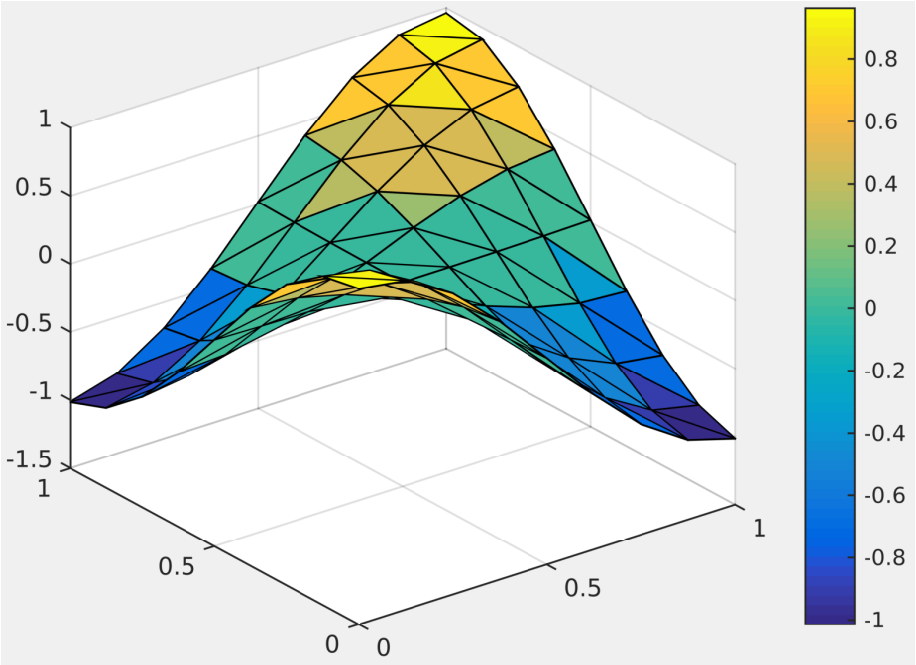

Figure 3: $\mathbb{P}_1$ approximate solution.

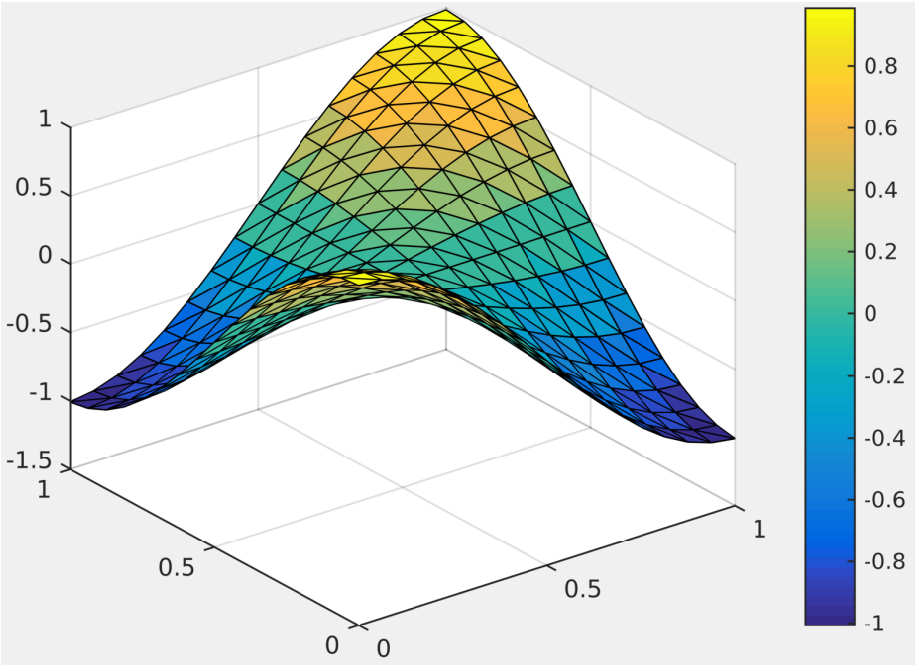

Figure 4: $\mathbb{P}_1$ approximate solution.

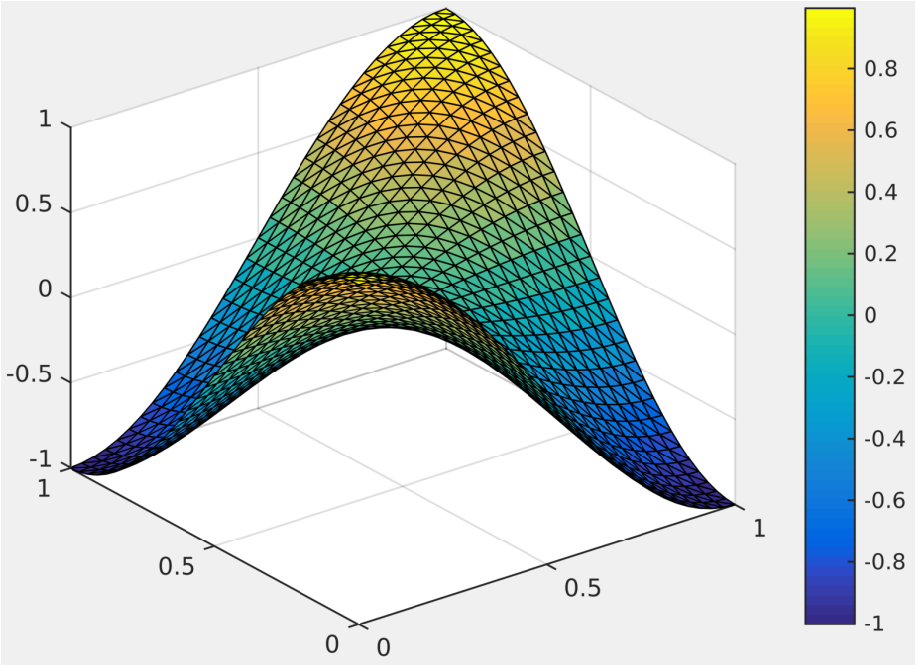

Figure 5: $\mathbb{P}_1$ approximate solution.

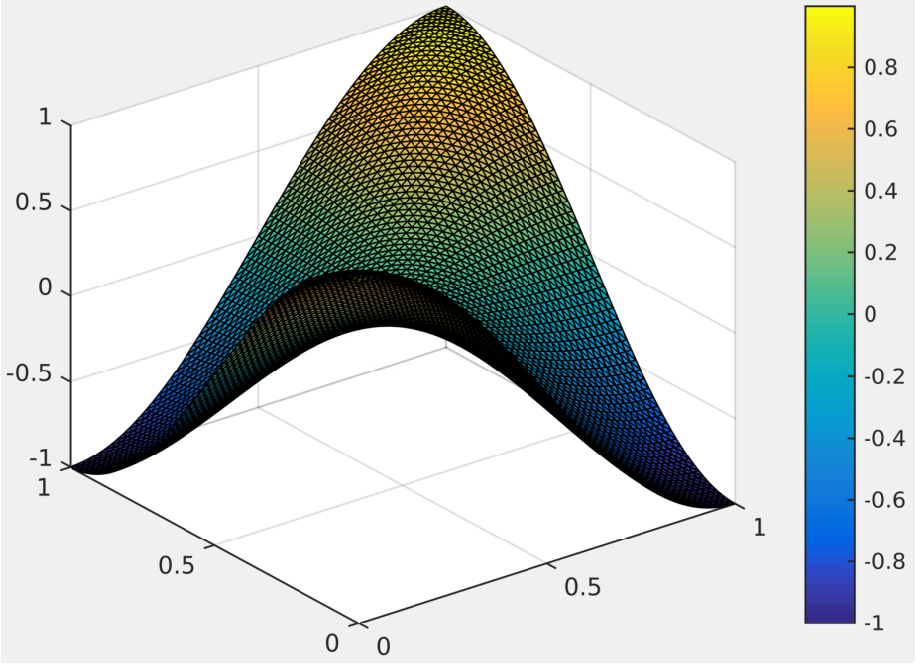

Figure 6: $\mathbb{P}_1$ approximate solution.

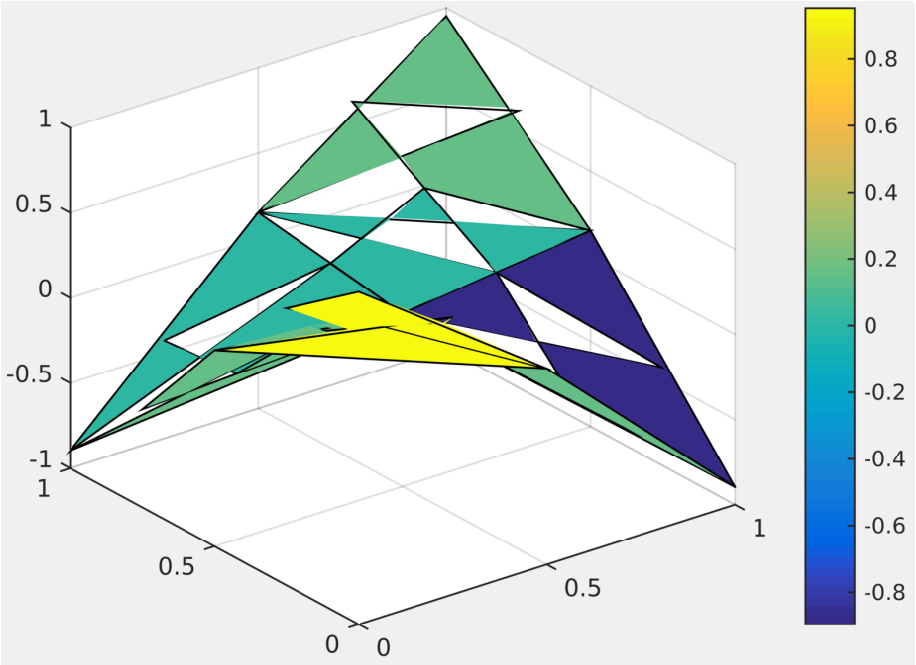

Figure 7: $\mathbb{P}_2$ approximate solution.

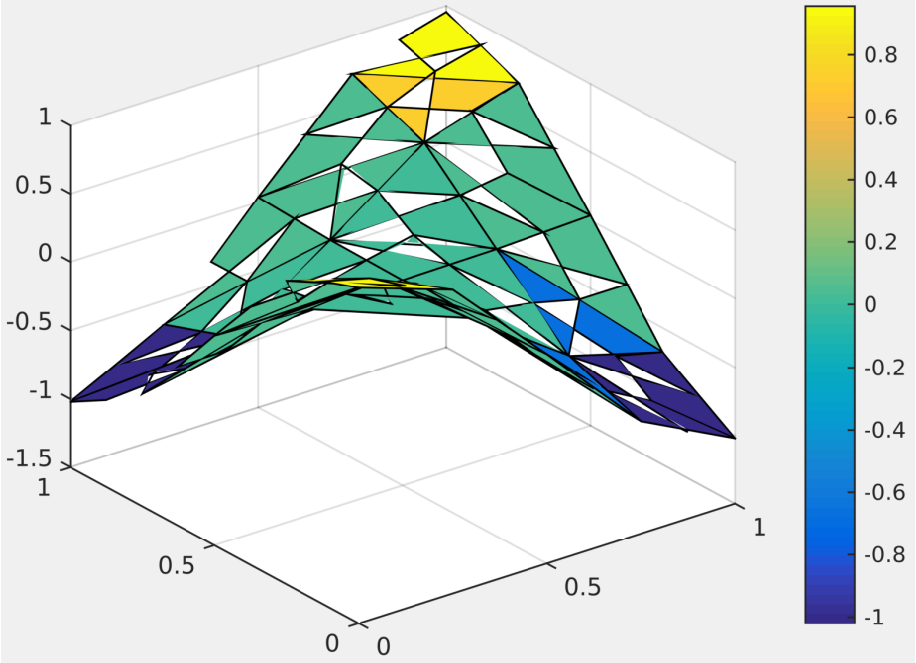

Figure 8: $\mathbb{P}_2$ approximate solution.

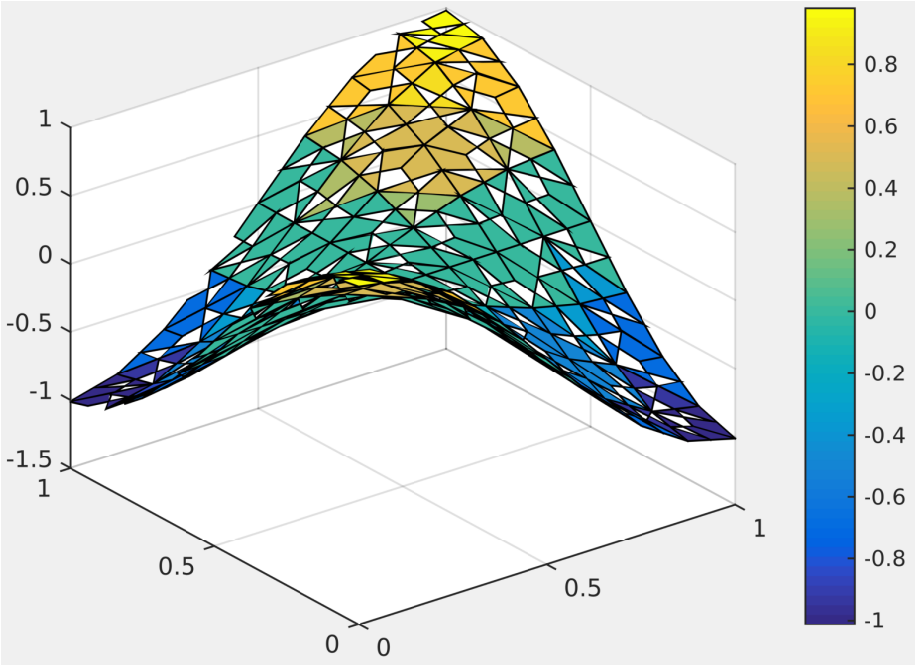

Figure 9: $\mathbb{P}_2$ approximate solution.

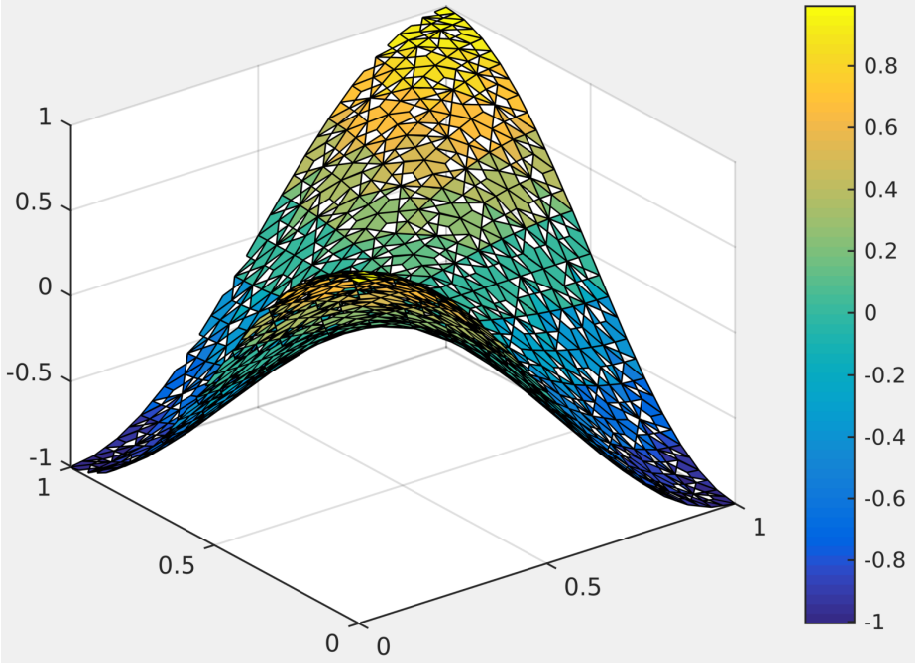

Figure 10: $\mathbb{P}_2$ approximate solution.

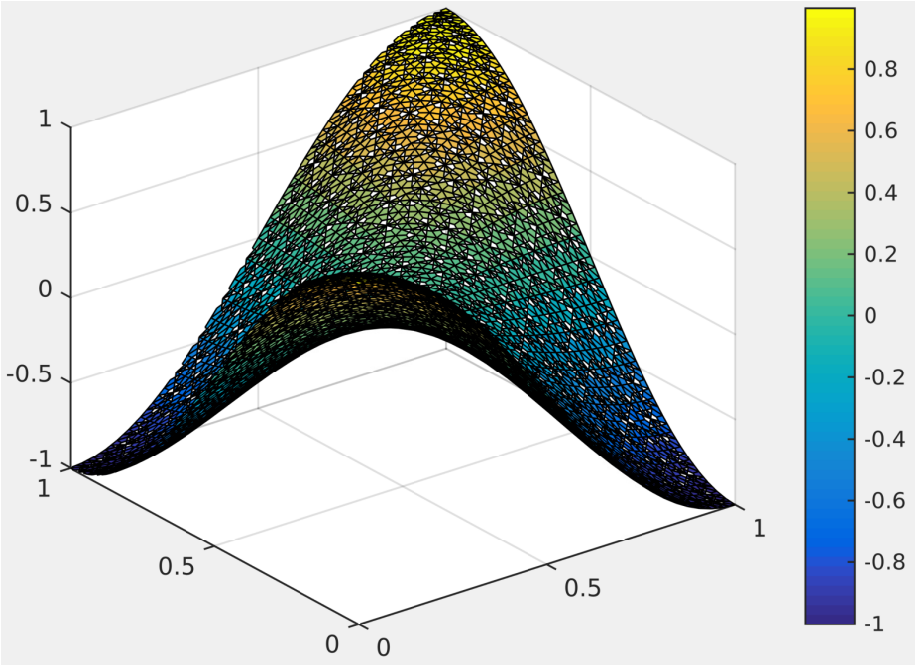

Figure 11: $\mathbb{P}_2$ approximate solution.

This completes our solution. □

Problem 2.2. *Using the function defined in Problem 2.1, write a script which calculates the numerical solution for different values of h and plot the curves of the relative errors for the euclidian and the infinity norms, with respect to h, in "loglog" scale:*

(a) for the $\mathbb{P}_1$ approximation.

(b) for the $\mathbb{P}_2$ approximation.

Solution. See Sec. 3.5. Running the following command lines in the main script

```
convergence_order(mesh_geo, 4, 1);
convergence_order(mesh_geo, 4, 2);
```

gives us

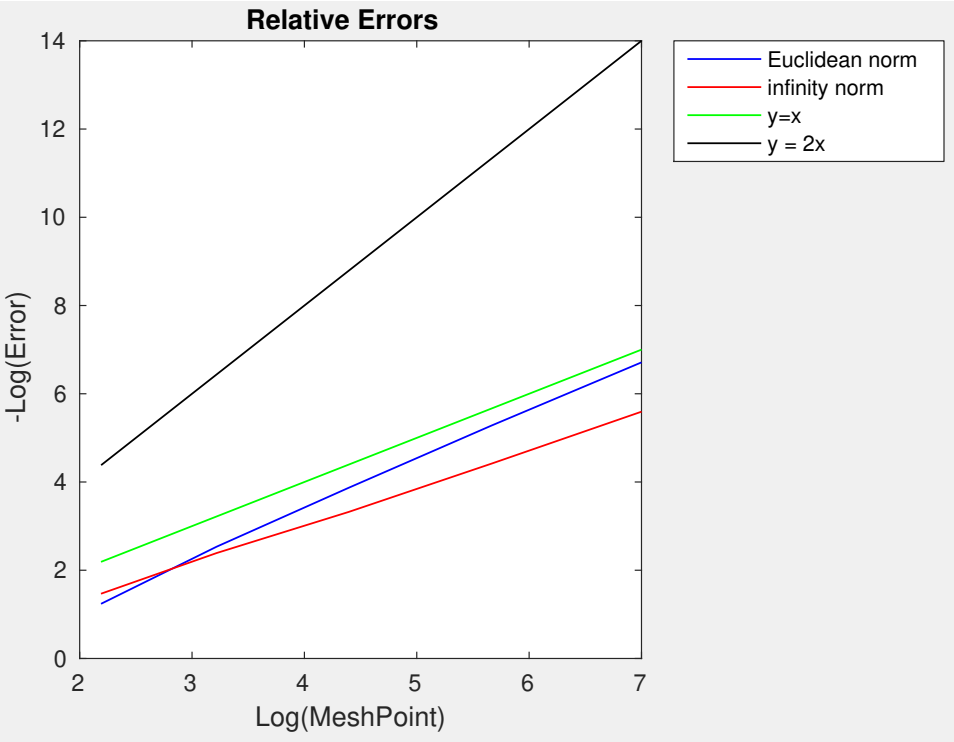

Figure 12: Convergence order of the $\mathbb{P}_1$ approximation

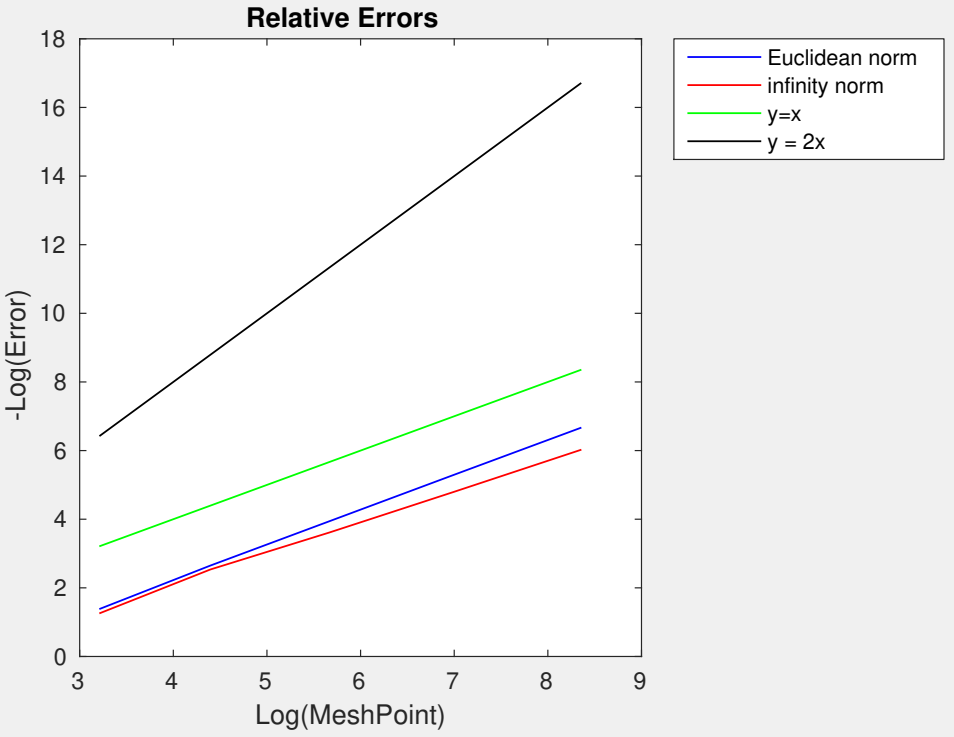

Figure 13: Convergence order of the $\mathbb{P}_2$ approximation

This completes our solution. □

3 Appendices: MATLAB Scripts

3.1 Function One

The following MATLAB script returns the function $f(x) = 1$.

```
function a = f_one(x)
% Evaluation of the function f(x) = 1 for all x
% Input:
% + x: a 1x2 or 2x1 vector
% Output:
% + a: the value 1

a = 1;
```

3.2 Exact Solution

The following MATLAB script returns the values of the proposed exact solution

$$u(x_1, x_2) = \cos(\pi x_1)\cos(\pi x_2), \quad \forall (x_1, x_2) \in [0,1]^2. \tag{3.1}$$

```
function exact_sol = exact_solution(x,y)
% The exact solution is given by u(x) = cos(pi*x)*cos(pi*y)
% Input:
% + x: a 2x1 or 1x2 vector
% Output:
% + exact_sol: the value of the exact solution at the point x, i.e.
% u(x,y) = cos(pi*x)*cos(pi*y)

exact_sol = cos(pi*x).*cos(pi*y);
```

3.3 Source Function

The following MATLAB script returns the values of the proposed source function

$$f(x_1, x_2) = \left(2\pi^2 + 1\right)\cos(\pi x_1)\cos(\pi x_2), \quad \forall (x_1, x_2) \in [0,1]^2. \tag{3.2}$$

```
function f = source_function(x)
% The source function is given by f = - Laplacian(u) + u, where u is the
% exact solution
% Input:
% + x: a 2x1 or 1x2 vector
% Output:
% + f: the value of the source function at the point x, i.e.
% f(x1,x2) = (2*pi^2 + 1)*cos(pi*x1)*cos(pi*x2)

f = (2*pi^2 + 1)*cos(pi*x(1))*cos(pi*x(2));
```

3.4 Solve the Neumann Boundary Value Problem

The following MATLAB script is used to solve the introduced Neumann BVP.

```
function [X, nodes] = solve_Neumann_BVP(mesh_geo, h, degree_FE)
% Solve the problem: -Laplacian(u) + u = f with the homogeneous Neumann
% boundary condition, on a domain described geometrically by mesh_geo, by
% application of the finite-element method (isoparametric P_1 or P_2). This
% function builds the mesh (P_1 or P_2) according to the discretization
% parameter h, then assemblies the matrix of the variational approximation,
```

```
% and finally solves the linear system.
% Author: Nguyen Quan Ba Hong
% Date: 8/11/2018
% Last Update: 8/11/2018
% Inputs:
% + mesh_geo: a mesh structure defining the geometry, if degree_FE = 1,
% this mesh structure must be a P_1 mesh structure, if degree_FE = 2, it
% must be modified into a P_2 mesh structure
% + h: discretization parameter of the mesh
% + degree_FE: degree of the finite-element interpolation
% Outputs:
% + X: a vector which contains the approximation of the exact solution at
% the nodes of the mesh
% + nodes: the coordinates of the nodes of the built mesh

%% Refine the Mesh Given
while 1
    % Compute the Mesh Size of the Current Mesh
    [~, max_edge_size] = find_worst_triangle(mesh_geo);
    if (max_edge_size <= h) % Stopping Criteria
        break % The Current Mesh is Fine Enough
    end
    mesh_geo = refine_mesh(mesh_geo);
end

%% Choose Mesh Structure According to the Degree of the FE Interpolation
if (degree_FE == 2) % The FE Interpolation of Second Order
    mesh_geo = P1_to_P2(mesh_geo); % Modify mesh_geo into P_2 Mesh Structure
end

%% Assembling Procedures
% Compute the Mass Matrix
M =  assemble_matrix('Id', 'Id', 'f_one', mesh_geo, degree_FE, 2);
% Compute the Stiffness Matrix
R1 = assemble_matrix('D1', 'D1', 'f_one', mesh_geo, degree_FE, 1);
R2 = assemble_matrix('D2', 'D2', 'f_one', mesh_geo, degree_FE, 1);
% Assembly the Finite-Element Matrix
A = M + R1 + R2;
% Assembly the Finite-Element Right Hand Side Vector
b = assemble_vector('Id', 'source_function', mesh_geo, degree_FE, 2);

%% Resolution
```

```
X = A\b; % Solve the Linear System Ax = b
nodes = mesh_geo.coords; % the Nodes of the Current Mesh
```

3.5 Convergence Order

The following MATLAB script is used to validate the convergence order of the finite-element method.

```
function [] = convergence_order(mesh_geo, times, degree_FE)
% Calculate the numerical solution for different values of the
% discretization parameter h and plot the curves of the relatives errors
% for the euclidean and the infinity norms, with respect to h, in "loglog"
% scale
% (a) for the P_1 approximation
% (b) for the P_2 approximation
% Author: Nguyen Quan Ba Hong
% Date: 8/11/2018
% Last Update: 8/11/2018
% Inputs:
% + mesh_geo: a mesh structure defining the geometry, if degree_FE = 1,
% this mesh structure must be a P_1 mesh structure, if degree_FE = 2, it
% must be modified into a P_2 mesh structure
% + times: the number of times which we refine the mesh given
% + degree_FE: degree of the finite-element interpolation
% + Output: a plot of the curves of the relative errors for the euclidean
% and the infinity norms, with respect to h, in "loglog" scale

%% Initializations
h_max = 100; % Fixed Maximal Discretization Parameter
relative_error_euclidean = zeros(times + 1,1);
relative_error_infinity = zeros(times + 1,1);
number_mesh_point = zeros(times + 1,1);

% Solve the Neumann BVP for the Initial Mesh
[X, nodes] = solve_Neumann_BVP(mesh_geo, h_max, degree_FE);
% Compute the Number of Nodes ofthe Initial Mesh
number_mesh_point(1) = size(nodes,1);
% Compute the Exact Solution on the Initial Mesh
U_ex = zeros(size(nodes,1),1); % Initialize
for i = 1:size(nodes,1)
    U_ex(i) = exact_solution(nodes(i,1), nodes(i,2));
end
```

```
% Compute the Relative Error for the Euclidean Norm
relative_error_euclidean(1) = norm(X - U_ex,2)/norm(U_ex,2);
% Compute the Relative Error for the infinity Norm
relative_error_infinity(1) = norm(X - U_ex,Inf)/norm(U_ex,Inf);

%% Main Loop
for i = 1:times % Refine the Inputted Mesh times Times
    mesh_geo = refine_mesh(mesh_geo); % Refine the Current Mesh
    [X, nodes] = solve_Neumann_BVP(mesh_geo, h_max, degree_FE);
    % Compute the Number of Nodes of the Current Mesh
    number_mesh_point(i+1) = size(nodes,1);
    % Compute the Exact Solution on the Initial Mesh
    U_ex = zeros(size(nodes,1),1); % Initialize
    for j = 1:size(nodes,1)
        U_ex(j) = exact_solution(nodes(j,1), nodes(j,2));
    end
    % Compute the Relative Error for the Euclidean Norm
    relative_error_euclidean(i+1) = norm(X - U_ex,2)/norm(U_ex,2);
    % Compute the Relative Error for the infinity Norm
    relative_error_infinity(i+1) = norm(X - U_ex,Inf)/norm(U_ex,Inf);
end

%% Plot the Relative Errors in the "loglog" Scale
figure
plot(log(number_mesh_point), -log(relative_error_euclidean), 'blue',...
     log(number_mesh_point), -log(relative_error_infinity), 'red',...
     log(number_mesh_point), 2*log(number_mesh_point), 'green');
xlabel('Log(MeshPoint)');ylabel('-Log(Error)');
title('Relative Errors');
legend('Euclidean norm','infinity norm', 'y = 2x','Location','NorthEastOutside');
```

References

[1] https://perso.univ-rennes1.fr/eric.darrigrand-lacarrieu/Teaching/M2RFEorganisation.html