Notes ⋆ Analyse Numérique - Part II: Implementation of the Finite Elements ⋆ Finite-Element Calculation

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 & their analysis are described in one & 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`
Copyright © 2016-2018 by Nguyen Quan Ba Hong. This document may be copied freely for the purposes of education and non-commercial research. Visit my site to get more.

Contents

1 Mathematical Framework

We consider the elliptic problem of order 2, written in variational form

$$\forall v \in V, \quad a(u,v) = l(v). \tag{1.1}$$

The bilinear form a & the linear form l involve integrals of the form

$$\int_\Omega \alpha(x)\, A(u(x), Du(x))\, B(v(x), Dv(x))\, dx \text{ or } \int_\Omega \alpha(x)\, F(v(x), Dv(x))\, dx, \tag{1.2}$$

(some integrals on the boundary $\partial\Omega$ may also be involved) where A, B, & F are linear operators. For example, the Laplace problem $-\Delta u = f$ leads to

$$\int_\Omega A_1(u, Du)\, B_1(v, Dv) + \int_\Omega A_2(u, Du)\, B_2(v, Dv) = \int_\Omega \alpha F(v, Dv), \tag{1.3}$$

with

- $A_1(w(x), Dw(x)) = B_1(w(x), Dw(x)) = \partial_{x_1} w(x)$,
- $A_2(w(x), Dw(x)) = B_2(w(x), Dw(x)) = \partial_{x_2} w(x)$,
- $\alpha(x) = f(x)$,
- $F(w(x), Dw(x)) = w(x)$.

On a practical point of view, if $(\varphi_1, \ldots, \varphi_N)$ is a basis of the discrete space of approximation V_h, we have to calculate the quantities

$$\int_\Omega \alpha(x)\, A(\varphi_J(x), D\varphi_J(x))\, B(\varphi_I(x), D\varphi_I(x))\, dx \text{ or } \int_\Omega \alpha(x)\, F(\varphi_I(x), D\varphi_I(x))\, dx. \tag{1.4}$$

To this aim, we split the integrals on the elements of the triangulation: for the first one

$$\sum_{K \in T_h} \int_K \alpha(x)\, A(\varphi_J(x), D\varphi_J(x))\, B(\varphi_I(x), D\varphi_I(x))\, dx, \tag{1.5}$$

& each integral on one element K is changed to an integral on the reference element $\widehat{K}$ by a change of variable $x = F_K(\widehat{x})$.

2 Quadrature Formulas

The calculation of an integral over the reference element is defined by a quadrature formula

$$\int_{\widehat{K}} f(\widehat{x})\, d\widehat{x} = \sum_{q=0}^{Q} w_q f(\widehat{x}_q). \tag{2.1}$$

The point $\widehat{x}_q$ & the weights w_q are given once at the beginning of the finite-element procedure. We give here, as an example, a nodal formula which is exact for $\mathbb{P}_1$ functions, & a nodal formula exact for $\mathbb{P}_3$ functions using the node of a $\mathbb{P}_2$ interpolation plus the iso-barycenter of the triangle, i.e.,

$$\int_{\widehat{K}} f\left(\widehat{x}\right) d\widehat{x} = \frac{1}{6}\left(f\left(0,0\right) + f\left(1,0\right) + f\left(0,1\right)\right), \quad \forall f \in \mathbb{P}_1\left(\widehat{K}\right), \tag{2.2}$$

$$\begin{aligned}\int_{\widehat{K}} f\left(\widehat{x}\right) d\widehat{x} = {} & \frac{1}{40}\left(f\left(0,0\right) + f\left(1,0\right) + f\left(0,1\right)\right) \\ & + \frac{1}{15}\left(f\left(\frac{1}{2},0\right) + f\left(\frac{1}{2},\frac{1}{2}\right) + f\left(0,\frac{1}{2}\right)\right) + \frac{9}{40} f\left(\frac{1}{3},\frac{1}{3}\right), \quad \forall f \in \mathbb{P}_2\left(\widehat{K}\right).\end{aligned} \tag{2.3}$$

Problem 2.1. *Write a* MATLAB *function which defines the two quadrature formulas. Its head is*

```
function [points, weights] = quadrature(number)
```

where the entry `number` *is an arbitrary number used to reference the requested formula. The output vectors are respectively of size* $(Q+1) \times 2$ *&* $(Q+1) \times 1$.

Solution. It is straightforward to carry out the (nodal) quadrature formulas defined by (2.2) and (2.3), see Sec. 8.1. □

We also give a MATLAB script which defines some Gaussian quadrature formulas. The high precision of these formulas will help us approximate the finite-element right hand side vector (in the assembling procedures) better.

3 Basis Functions

For the elementary calculation, we use only the basis functions of the reference element. The basis functions are then defined on the effective element K thanks to the geometric function F_K.

Problem 3.1. *Write a* MATLAB *function which evaluates the nodal basis functions* $\mathbb{P}_k$ $(k = 1, 2)$ *of the reference element at a given point:*

```
function [vals, dervals] = shape_fcts(xh, k)
```

The entries are `xh`*, the point where we want to evaluate the functions, and* `k`*, degree of the finite element. The output is a table* `vals` *of the values of the basis functions at* `xh` *(table of size* $NbRefNodes \times 1$*, in the order of the local numbering of the reference element, where* $NbRefNodes$ *denotes the number of nodes on the reference element), & the table* `dervals` *of the values of the corresponding derivatives in both directions (table of size* $NbRefNodes \times 2$*).*

Solution. The nodal basis functions $\mathbb{P}_1$ of the reference element are given by $\widehat{\lambda}_1(\widehat{x}) = \widehat{x}_1$, $\widehat{\lambda}_2(\widehat{x}) = \widehat{x}_2$, $\widehat{\lambda}_3(\widehat{x}) = 1 - \widehat{x}_1 - \widehat{x}_2$ and the nodal basis functions basis functions $\mathbb{P}_2$ of the reference element are given by $\widehat{\varphi}_i = \widehat{\lambda}_i \left(2\widehat{\lambda}_i - 1\right)$ and $\widehat{\varphi}_{i+3} = 4\widehat{\lambda}_i\widehat{\lambda}_{i+\text{ mod }(i,3)}$, for all $i = 1, 2, 3$. It is straightforward to evaluate these nodal basis functions and their gradients, see Sec. 8.3. □

4 Geometric Calculation

The elementary integrals require the evaluation of the quantities $\varphi_I(x)$ & $\partial_{x_l}\varphi_I(x)$. They are expressed from the finite-element interpolation basis functions of the reference element, & the geometric function $x = F_K(\widehat{x})$. If the vertex i of the effective element K has the global number I, we have the relations

- $\varphi_I(x) = \widehat{\varphi}_i(\widehat{x})$,
- $\partial_{x_l}\varphi_I(x) = \sum_{k=1}^{2} \partial_{\widehat{x}_k}\widehat{\varphi}_i(\widehat{x}) \left[JF_K(\widehat{x})^{-1}\right]_{k,l}$, or equivalently $\nabla_x\varphi_I(x) = JF_K(\widehat{x})^{-T}\nabla_{\widehat{x}}\widehat{\varphi}_i(\widehat{x})$.

Moreover, the Jacobian matrix of the geometric function F_K is evaluated thanks to the geometric basis functions (= shape functions) $\widehat{\psi}_i$:

$$[JF_K(\widehat{x})]_{k,l} = \sum_{i=1}^{n_{\text{geo}}} (V_i)_k \partial_{\widehat{x}_l}\widehat{\psi}_i(\widehat{x}), \tag{4.1}$$

where V_i are the nodes (the vertices in $\mathbb{P}_1$) of the effective element K.

Problem 4.1. *Write a* MATLAB *function which evaluates the Jacobian matrix of the geometric function $F_K(\widehat{x})$ at the point $\widehat{x}$:*

```
function jacob = compute_jacobian(vertices, dervals_psi)
```

The entries are the coordinates of the nodes `vertices` *& the values* `dervals_psi` *of the derivatives of the geometric basis functions at the point $\widehat{x}$.*

Solution. It is straightforward to carry out (4.1) into a script, see Sec. 8.4. □

5 Assembling Procedure

The assembling procedure consists in the calculation of a matrix of the form

$$\int_\Omega \alpha(x) A(\varphi_J(x), D\varphi_J(x)) B(\varphi_I(x), D\varphi_I(x)) dx, \tag{5.1}$$

by a summation of all the elementary contributions. The scheme of this procedure is the following:

```
Loop on the elements of the mesh
   Loop on the quadrature points
      Geometric calculation at point xq.
      Loop on the nodes i of the element K
         I = global number of i
         Loop on the nodes j of the element K
            J = global number of j
            A(I,J) = A(I,J) + contribution of (i,j) at xq
         end
      end
   end
end
```

Problem 5.1. *Write a* MATLAB *function which assemblies the finite-element matrix. Its head is*

```
function A = assemble_matrix(str_integrand_unknown, str_integrand_test,
                             ... str_cofvar, mesh_geo, degree_FE, number)
```

The entries are:

- `str_integrand_unknown`, `str_integrand_test`*: character strings identifying the operators A and B (see function* `diff_op.m` *in annex),*
- `str_cofvar`*: character string identifying the function $\alpha(x)$,*
- `mesh_geo`*: mesh structure defining the geometry,*
- `degree_FE`*: degree of the finite-element interpolation,*
- `number`*: identification number of the quadrature formula.*

A function `assemble_vector` *will be written for the calculation of the finite-element right hand side, on the same model.*

Solution. See Sec. 8.5 and 8.6. □

6 Essential Conditions (Optional)

To take in consideration the essential conditions (Dirichlet) one requires the identification of the nodes (or edges) of the boundary of the domain where the condition is imposed.

Problem 6.1. *Using the function* `build_edge_connectivity` *from the mesh-design exercises, build the set of edges defining the subdomain $\partial\Omega$.*

Proof. See Sec. 8.7 and 8.8. □

Problem 6.2. *Write a procedure* elimination*, which consists of - for a given node I on the Dirichlet boundary - vanishing the I^{th} component of the right hand side and all the entries of the I^{th} line of the matrix except the diagonal one. You may choose the head function*

```
function [Aout, Bout] = Dirichlet_elimination(Ain, Bin, boundary_nodes)
```

The table entry `boundary_nodes` *contains the global numbers of the nodes on the Dirichlet boundary.*

7 Resolution and Post-Treatment

For the resolution and the post-treatment, you may use the following MATLAB functions

- `A\b` to solve the linear system $Ax = b$,
- `trisurf(triangles, coords(:,1), coords(:,2), X);` to plot the solution X at the node of the mesh.

8 Appendices: MATLAB Scripts

8.1 Nodal Quadrature Formulas

The following MATLAB script is used to define the two (nodal) quadrature formulas given by (2.2)-(2.3). These quadrature formulas are used to calculate an integral over the reference element.

```
function [points, weights] = quadrature(number)
% Define the following two quadrature formulas to calculate an integral over
% the reference element
% number = 1: a nodal formula which is exact for all P_1 functions
% number = 2: a nodal formula which is exact for all P_3 functions by using
% the nodes of a P_2 interpolation plus the iso-barycenter of the triangle
% Author: Nguyen Quan Ba Hong
% Date: 12/10/2018
% Last Update: 12/10/2018
% Input:
% + number: take values 1 or 2, indicate which nodal formula will be used
% Outputs:
% + points: corresponding nodes in the reference element used for
% the nodal formula chosen
% + weights: corresponding weights used for the nodal formula chosen

if (number == 1) % P_1 Quadrature Formula
    points = [1,0; 0,1; 0,0]; % Nodes
    weights = [1/6; 1/6; 1/6]; % Weights
elseif (number == 2) % P_3 Quadrature Formula
    points = [0,0; 1,0; 0,1; .5,0; .5,.5; 0,.5; 1/3,1/3]; % Nodes
    weights = [1/40; 1/40; 1/40; 1/15; 1/15; 1/15; 9/40]; % Weights
else
    disp('Invalid Input');
end
```

8.2 Gaussian Quadrature Formulas

The following MATLAB script is used to define some Gaussian quadrature formulas.

```
function [points, weights] = Gaussian_quadrature(number)
% Define some Gaussian quadrature formulas to calculate an integral over
% the reference triangle
% Author: Nguyen Quan Ba Hong
% Date: 7/11/2018
% Last Update: 7/11/2018
% Input:
% + number: take values 1, 2, 3 or 4, indicate which Gaussian quadrature
% formula will be used
% Outputs:
% + points: the corresponding Gaussian quadrature nodes in the reference
% element used for the Gaussian quadrature formula chosen
% + weights: the corresponding Gaussian quadrature weights used for the
% Gaussian quadrature formula chosen

switch number
    case 1
        weights = 1;
        points = [1/3 1/3];
    case 2
        weights = [1/3 1/3 1/3];
        points = [1/6 1/6;
            2/3 1/6;
            1/6 2/3];
    case 3
        weights = [-27/48 25/48 25/48 25/48];
        points = [1/3 1/3;
            0.2 0.2;
            0.6 0.2;
            0.2 0.6];
    case 4
        weights = [0.223381589678011
            0.223381589678011
            0.223381589678011
            0.109951743655322
            0.109951743655322
            0.109951743655322];
        points = [0.445948490915965 0.445948490915965;
            0.445948490915965 0.108103018168070;
            0.108103018168070 0.445948490915965;
            0.091576213509771 0.091576213509771;
            0.091576213509771 0.816847572980459;
            0.816847572980459 0.091576213509771];
    otherwise
        error('Invalid Input')
end
```

8.3 Evaluate Nodal Basic Functions

The following MATLAB script is used to evaluate the values and the derivatives of the nodal basis functions $\mathbb{P}_k$ $(k = 1, 2)$ of the reference element $\widehat{K}$ at a given point.

```
function [vals, dervals] = shape_fcts(xh,k)
% Evaluate the nodal basis functions P_k (k = 1,2) of the reference element
% at a given point
% Author: Nguyen Quan Ba Hong
% Date: 12/10/2018
% Last Update: 12/10/2018
% Inputs:
% + xh: the point where we want to evaluate the functions
% + k: the degree of the finite element, k = 1 or 2
% Outputs:
% + vals: a table of the values of the basis functions at the point xh, size:
% NbRefNodes x 1
% + dervals: a table of the values of the corresponding derivatives in both
% directions, size: NbRefNodes x 2

if (k == 1) % First-Order Finite Element
```

```
    vals = [xh(1); xh(2); 1 - xh(1) - xh(2)]; % Values of Nodal Basis Functions P_1
    dervals = [1,0; 0,1; -1,-1]; % Derivatives of Nodal Basis Function P_1
elseif (k == 2) % Second-Order Finite Element
    [a, b] = shape_fcts(xh, 1); % Reuse the Nodal Basis Functions P_1
    vals = a.*(2*a - 1); % First 3 Nodal Basis Functions P_2
    % Add 3 Nodal Basis Functions P_2
    vals = [vals; 4*a(1)*a(2); 4*a(2)*a(3); 4*a(3)*a(1)];
    % Derivatives of Nodal Basis Functions P_2
    dervals = [(4*a(1) - 1)*b(1,:); (4*a(2) - 1)*b(2,:);
        (4*a(3) - 1)*b(3,:); 4*(a(2)*b(1,:) + a(1)*b(2,:));
        4*(a(2)*b(3,:) + a(3)*b(2,:)); 4*(a(3)*b(1,:) + a(1)*b(3,:))];
else
    disp('Invalid Input');
end
```

8.4 Jacobian Matrix of the Geometric Function

The following MATLAB script is used to evaluate the Jacobian matrix of the geometric function $F_K(\widehat{\mathbf{x}})$ at the point $\widehat{\mathbf{x}}$.

```
function jacob = compute_jacobian(vertices, dervals_psi)
% Evaluate the Jacobian matrix of the geometric function at a given point
% Author: Nguyen Quan Ba Hong
% Date: 15/10/2018
% Last Update: 15/10/2018
% Inputs:
% + vertices: the coordinates of the main (in P_1) nodes of a triangle, size: 3x2
% + dervals_psi: the values of the derivatives of the geometric basis
% functions at the point hat(x)
% Output:
% + jacob: the value of Jacobian matrix of the geometric function F_K(hat(x))
% at the given point hat(x)

jacob = zeros(2); % Initialize
for k = 1:2
    for l = 1:2
        jacob(k,l) = dot(vertices(:,k), dervals_psi(:,l));
    end
end
```

8.5 Assembly the Finite-Element Matrix

The following MATLAB script is used to assembly the finite-element matrix.

```
function A = assemble_matrix(str_integrand_unknown, str_integrand_test, ...
    str_cofvar, mesh_geo, degree_FE, number)
% Assembly the finite-element matrix
% Author: Nguyen Quan Ba Hong
% Date: 15/10/2018
% Last Update: 6/11/2018
% Inputs:
% + str_integrand_unknown, str_integrand_test: character strings
% identifying the operators A & B (see function diff_op.m)
% + str_cofvar: character string identifying the function alpha(x)
% + 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, before running this script
% + degree_FE: degree of the finite-element interpolation
% + number: identification number of the quadrature formula
% Output:
% + A: the assembled finite-element matrix

%% Points & Weights of Quadrature Formulas for the Reference Element
[points, weights] = quadrature(number); % Nodal Quadrature Formulas
% [points, weights] = Gausspoints(2); % Gaussian Quadrature Formulas

%% Main Loops
A = zeros(size(mesh_geo.coords,1)); % Initialzie
for K = 1:size(mesh_geo.triangles,1) % Loop on All Triangles of the Mesh Given
    % Coordinates of the Vertices of the K-th Triangle in the Mesh
    % the Vertices in P_K of this Triangle
    vertices = mesh_geo.coords(mesh_geo.triangles(K,:),:);
    % the Vertices in P_1 of this Triangle
    main_vertices = vertices([1,2,3],:);
    for q = 1:size(points,1) % Loop on All Quadrature Points
        % Evaluate the Nodal Basis Functions of the Reference Element at
        % the Quadrature Points
        [vals_hatphi, dervals_hatphi] = shape_fcts(points(q,:), degree_FE);
        % Evaluate the Geometric Basis Functions at the Quadrature Points
        [vals_hatpsi, dervals_hatpsi] = shape_fcts(points(q,:), 1); %P_1 Isoparametric
        % Evaluate the Geometric Function at the Quadrature Point
        geo_func = [dot(vals_hatpsi, main_vertices(:,1)), ...
```

```
            dot(vals_hatpsi, main_vertices(:,2))];
        % Evaluate the Jacobian Matrix of the Geometric Function at the
        % Quadrature Point
        jacob = compute_jacobian(main_vertices, dervals_hatpsi);
        % Compute the Gradient of the Function phi(x)
        dervals_phi = dervals_hatphi/jacob;
        for i = 1:size(mesh_geo.triangles,2) % Loop on the Node i of the Element K
            % Global Index of the Node i of the Element K
            I = mesh_geo.triangles(K, i);
            for j = 1:size(mesh_geo.triangles,2) % Loop on the Node j of the Element K
                % Global Index of the Node j of the Element K
                J = mesh_geo.triangles(K, j);
                % Add Contribution of (i,j) at the Quadrature Points
                A(I,J) = A(I,J) + weights(q)*feval(str_cofvar, geo_func)*...
                    diff_op(str_integrand_unknown, vals_hatphi(j), dervals_phi(j,:))*...
                    diff_op(str_integrand_test, vals_hatphi(i), dervals_phi(i,:))...
                    *abs(det(jacob));
            end
        end
    end
end
```

8.6 Assemble the Finite-Element Right Hand Side Vector

The following MATLAB script is used to assembly the finite-element right hand side vector.

```
function b = assemble_vector(str_integrand_test, str_cofvar, mesh_geo, degree_FE, number)
% Assembly the finite-element right hand side vector
% Author: Nguyen Quan Ba Hong
% Date: 15/10/2018
% Last Update: 6/11/2018
% Inputs:
% + str_integrand_test: character string identifying the operator B
% + str_cofvar: character string identifying the function alpha(x)
% + 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, before running this script
% + degree_FE: degree of the finite-element interpolation
% + number: identification number of the quadrature formula
% Output:
% + assemble_vector: assembled finite-element RHS vector
```

```
%% Points & Weights of Quadrature Formulas for the Reference Element
[points, weights] = quadrature(number); % Nodal Quadrature Formulas
% [points, weights] = Gaussian_quadrature(4); % Gaussian Quadrature Fofmulas

%% Main Loops
b = zeros(size(mesh_geo.coords,1),1); % Initialize
for K = 1:size(mesh_geo.triangles,1) % Loop on All Triangles of the Mesh Given
    % Coordinates of Vertices of the K-th Triangle in the Mesh
    % the Vertices in P_K of this Triangle
    vertices = mesh_geo.coords(mesh_geo.triangles(K,:),:);
    main_vertices = vertices([1,2,3],:); % the Vertices in P_1 of this Triangle
    for q = 1:size(points,1) % Loop on All Quadrature Points
        % Evaluate the Nodal Basis Functions of the Reference Element at
        % the Quadrature Points
        [vals_hatphi, dervals_hatphi] = shape_fcts(points(q,:), degree_FE);
        % Evaluate the Geometric Basis Functions at the Quadrature Points
        [vals_hatpsi, dervals_hatpsi] = shape_fcts(points(q,:), 1); %P_1 Isoparametric
        % Evaluate the Geometric Function at the Quadrature Point
        geo_func = [dot(vals_hatpsi, main_vertices(:,1)), ...
            dot(vals_hatpsi, main_vertices(:,2))];
        % Evaluate the Jacobian Matrix of the Geometric Function at the
        % Quadrature Point
        jacob = compute_jacobian(main_vertices, dervals_hatpsi);
        % Compute the Gradient of the Function phi(x)
        dervals_phi = dervals_hatphi/jacob;
        for i = 1:size(mesh_geo.triangles,2) % Loop on the Node i of the Element K
            % Global Index of the Node i of the Element K
            I = mesh_geo.triangles(K, i);
            % Add Contribution of (i,j) at the Quadrature Point
            b(I) = b(I) + weights(q)*feval(str_cofvar, geo_func)*...
                diff_op(str_integrand_test, vals_hatphi(i), dervals_phi(i,:))...
                *abs(det(jacob));
        end
    end
end
```

8.7 Boundary Edges

The following MATLAB script is used to build the set of edges defining the subdomain $\partial\Omega$.

```
function boundary_edges = boundary_edges(mesh)
% Build the set of edges defining the boundary of a given mesh
% Author: Nguyen Quan Ba Hong
% Date: 17/10/2018
% Last Update: 17/10/2018
% Input:
% + mesh: a P_1 mesh structure
% Output:
% + boundary_edges: the set of edges defining the boundary of the mesh
% given

%% Construct the Tables edges & edges_triangles of the Given Mesh
[edges, edges_triangles] = build_edge_connectivity(mesh);

%% Determine the Boundary of the Given Mesh
boundary_edges = []; % Initialize
for i = 1:size(edges,1) % Loop on All Edges of the Mesh Given
    if (size(edges_triangles{i},1) == 1) % This is a Boundary Edge
        % Store this Boundary Edge
        boundary_edges = [boundary_edges; edges(i,:)];
    end
end
```

8.8 Elimination Procedure for Dirichlet Boundary Conditions

The following MATLAB script is used to carry out the Dirichlet elimination procedure.

```
function [Aout, Bout] = Dirichlet_elimination(Ain, Bin, boundary_nodes)
% An elimination procedure, which consists of - for a given node I on the
% Dirichlet boundary - vanishing the I-th component of the right hand side
% and all entries of the I-th line of the matrix except the diagonal one.
% Author: Nguyen Quan Ba Hong
% Date: 17/10/2018
% Last Update: 17/10/2018
% Inputs:
% + Ain: the assembled matrix
% + Bin: the assembled vector
% + boundary_nodes: the table entry boundary_nodes contains the global
% numbers of the nodes on the Dirichlet boundary
% Outputs:
% + Aout: the assembled matrix after the Dirichlet elimination procedure
% + Bout: the assembled vector after the Dirichlet elimination procedure

for i = 1:size(boundary_nodes,1)
```

```
    % Vanish the i-th component of the right hand side
    Bin(i) = 0;
    % Vanish all the entries of the i-th line of the matrix except the diagonal one
    temp = Ain(i,i);
    Ain(i,:) = zeros(1,size(Ain,1)); % Vanish all entries of A(i,:)
    Ain(i,i) = temp; % Except the diagonal entry A(i,i)
end
Aout = Ain;
Bout = Bin;
```

8.9 Differential Operators

We give here the source code of the function `diff_op` which defines the differential operators involved in the variational formulation

```
function g = diff_op(type, u, Du)
% Differentiel operator
% Author: G.Vial
% Date: 16/09/2010
% Last update: 17/11/10
%
% usage: g = diff_op(type,u,Du)
%
% input:
%  type : type of operator (string)
%  u    : value of function (scalar)
%  Du   : value of function derivative (1 by 2 array)
%
% output:
% g : value of operator

switch (type)
   case 'Id'
      g=u;
   case 'D1'
      g=Du(1);
   case 'D2'
      g=Du(2);
   otherwise
      error(['Operator ',type,' not defined -- :-((( --'])
end
```

References

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