The Galerkin Finite Element Method: Implementation [1]

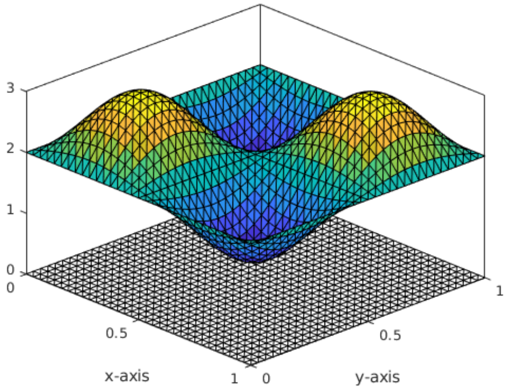
Alfio Quarteroni and Francesco Regazzoni

MOX, Dipartimento di Matematica
Politecnico di Milano

Notes for the course: Numerical Analysis of Partial Differential Equations
A.Y. 2019-2020

[1] Credits: P.F. Antonietti, I. Mazzieri

Implementation of linear finite elements on a fixed mesh



Poisson's problem:

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

Finite element formulation of the Poisson's problem (g=0)

Variational formulation

$$\text{Find } u \in H_0^1 \quad \text{s.t.} \quad a(u,v) = F(v) \quad \forall\, v \in H_0^1.$$

Galerkin formulation

$$\text{Find } u_h \in V_h \subset H_0^1 \quad \text{s.t.} \quad a(u_h,v_h) = F(v_h) \quad \forall\, v_h \in V_h$$

$$\begin{aligned} a(w,v) &= \int_\Omega \nabla w \cdot \nabla v = \sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \nabla w \cdot \nabla v \qquad \forall w, v \in H_0^1, \\ F(w) &= \int_\Omega f w = \sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} f w \qquad \forall\, w \in H_0^1. \end{aligned}$$

$\mathcal{T}_h$ triangulation of Ω made by triangles $\mathcal{K}$.

Algebraic formulation (P1 elements)

- Fix a basis for V_h, i.e.

$$V_h = \mathrm{span}\{\varphi_i,\ i = 1 \dots, N_h\},$$

where N_h denotes the total number of degrees of freedom in $\mathcal{T}_h$, $\varphi_i \in C^0(\mathcal{T}_h)$ and $\varphi_i \in \mathbb{P}^1(\mathcal{K})$ for any $\mathcal{K} \in \mathcal{T}_h$.

- Expand the discrete solution in terms of the basis, i.e.

$$u_h(\mathbf{x}) = \sum_{j=1}^{N_h} u_j \varphi_j(\mathbf{x})$$

- The discrete problem becomes: Find $\mathbf{u} = [u_1, u_2, \dots, u_{N_h}]^T \in \mathbb{R}^{N_h}$ s.t.

$$\sum_{j=1}^{N_h} u_j a(\varphi_j, \varphi_i) = F(\varphi_i) \quad \forall\, i = 1 \dots, N_h$$

Algebraic formulation (cont'd)

Algebraic formulation

$$\text{Find } \mathbf{u} \in \mathbb{R}^{N_h} \quad \text{s.t. } \mathbf{A}\mathbf{u} = \mathbf{b}$$

where

$$\mathbf{A}(i,j) = a(\varphi_j, \varphi_i) \qquad i,j = 1 \ldots, N_h$$
$$\mathbf{b}(i) = F(\varphi_i) \qquad i = 1 \ldots, N_h$$

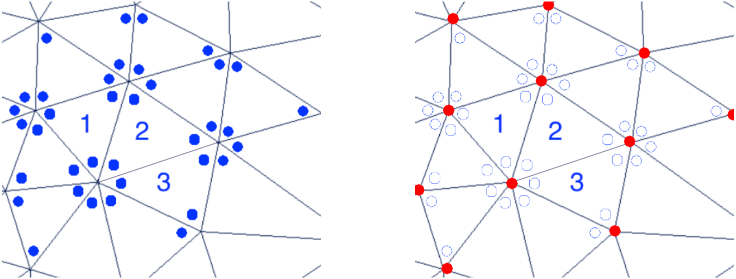
Implementation

We want a computer program that:

1. Reads a triangulation defining the domain
2. Assembles the system matrix and right-hand side vector
3. Solves the system and outputs the solution

1. Mesh generation

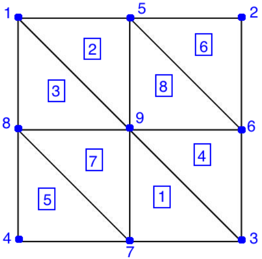
$\boxed{\text{Region} = \text{C\_create\_mesh(Dati);}}$



- For any triangle $\mathcal{K} \in \mathcal{T}_h$ there are $\mathrm{nln} = 3$ local degrees of freedom - dof (•)
- Global degrees of freedom (•) are obtained by imposing continuity constraints for the basis functions

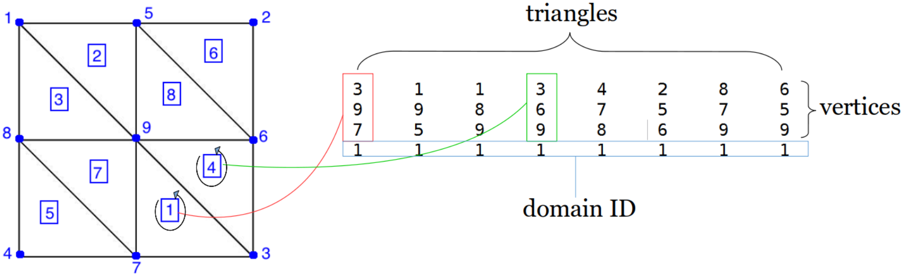
A quick look into the code: the **Region** structure

```
Region.dim       ->  problem dimension (2)
Region.MeshType -> (string) 'TU' unstructured or 'TS' structured triangular mesh
Region.domain   -> (2x2 matrix, real) domain limits
Region.h        -> mesh size
Region.nvert    -> number of vertices of the triangulation
Region.nel      -> number of elements
Region.coord_x  -> coordinates of the mesh nodes (x)
Region.coord_y  -> coordinates of the mesh nodes (y)
Region.coord    -> coordinates of the mesh nodes (x,y)
Region.boundary_edges -> connectivity of boundary edges
Region.connectivity   -> connectivity of the mesh triangles
```



A quick look into the code: the **Region** structure

```
Region.connectivity   -> connectivity of the mesh triangles
```



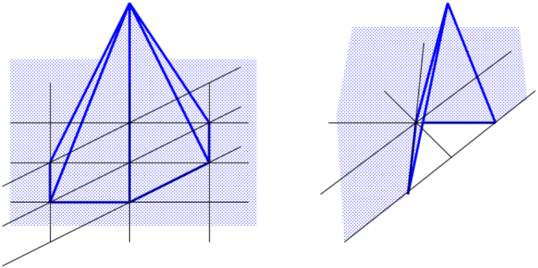
Note that the vertices of each triangle are listed in a "counter-clockwise" order.

Towards the implementation

Consider the nodal basis $\{\varphi_j\}_{j=1}^{N_h}$ of V_h of functions

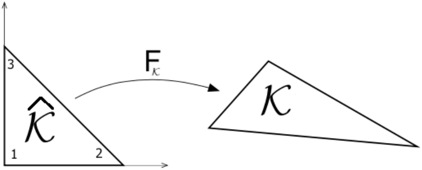
$$\varphi_j \in V_h : \qquad \varphi_j(\mathbf{x}_i) = \delta_{ij}, \qquad i,j = 1,2,...,N_h$$

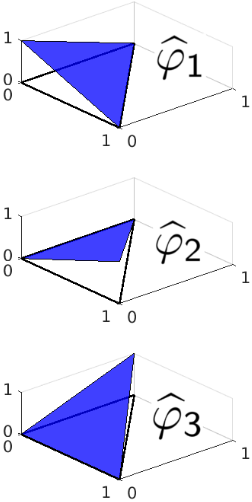
where $\mathbf{x}_i$, $i = 1,2,...,N_h$ are the vertices of the triangulation.



Then if $v \in V_h$, $v(x) = \sum_{j=1}^{N_h} v_j \varphi_j(x) = \sum_{j=1}^{N_h} v(x_j)\varphi_j(x)$

Linear shape functions on the reference triangle





$$\begin{cases} \widehat{\varphi}_1(\xi,\eta) = 1 - \xi - \eta & \text{node } (0,0) \to (x_1, y_1), \\ \widehat{\varphi}_2(\xi,\eta) = \xi & \text{node } (1,0) \to (x_2, y_2), \\ \widehat{\varphi}_3(\xi,\eta) = \eta & \text{node } (0,1) \to (x_3, y_3), \end{cases}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} x2 - x1 & x3 - x1 \\ y2 - y1 & y3 - y1 \end{pmatrix}}_{\mathbf{B}_{\mathcal{K}}} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \underbrace{\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}}_{\mathbf{b}_{\mathcal{K}}}$$

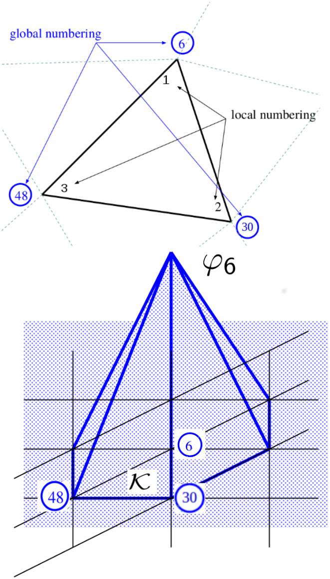
Then $\varphi_{j|_{\mathcal{K}}} = \widehat{\varphi}_j \circ {\mathbf{F}_{\mathcal{K}}}^{-1}$ and $\widehat{\varphi}_j = \varphi_j \circ \mathbf{F}_{\mathcal{K}}$.

Linear shape functions on the physical triangle

Observe that

$$
\begin{aligned}
\varphi_{6|\mathcal{K}} &= \widehat{\varphi_1}_{|\hat{\mathcal{K}}} \\
\varphi_{30|\mathcal{K}} &= \widehat{\varphi_2}_{|\hat{\mathcal{K}}} \\
\varphi_{48|\mathcal{K}} &= \widehat{\varphi_3}_{|\hat{\mathcal{K}}}
\end{aligned}
$$

where $\varphi_{j|\mathcal{K}}$ is the linear function on $\mathcal{K}$ that equals one at the j-th local vertex and zero at the others.



A quick look into the code: the **femregion** struct

$\boxed{\text{femregion} = \text{C\_create\_femregion(Dati,Region)}}$

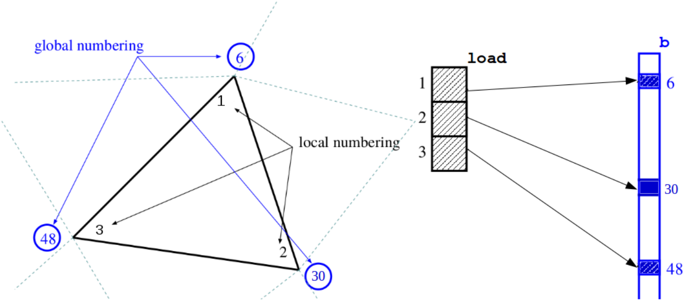
```
femregion.fem        -> 'P1' conforming linear finite elements
femregion.domain     -> Region.domain
femregion.type_mesh -> Region.MeshType
femregion.h          -> Region.h
femregion.nln        -> number of local degrees of freedom
femregion.ndof       -> number of global degrees of freedom
femregion.ne         -> Region.ne
femregion.dof        -> Region.coord
femregion.nqn_1D     -> Dati.nqn_1D (num. of quad. nodes for integral over lines)
femregion.nqn_2D     -> Dati.nqn_2D (num. of quad. nodes for integral over surface)
femregion.degree     -> degree (polynomial order)
femregion.coord      -> Region.coord
femregion.connectivity    -> Region.connectivity (local to global map)
femregion.boundary_points -> list of boundary points
                             (to be used for Dirichlet conditions)
```

2. Assembly of the right-hand side

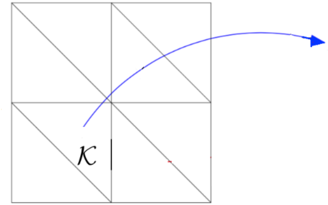
$$\mathbf{b}(i) = F(\varphi_i) = \int_\Omega f\varphi_i\,dx = \sum_{\mathcal{K}\in\mathcal{T}_h,\,\mathcal{K}\subset supp(\varphi_i)} \underbrace{\int_\mathcal{K} f\varphi_i\,dx}_{\text{quadrature formulas}} \qquad i = 1\ldots, N_h$$

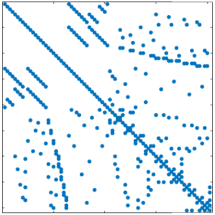
Idea: Loop over the elements $\mathcal{K}$, for each element do:

- compute all the integrals on the element (local vector **load**);
- add the computed integrals at the proper positions of the right-hand side vector **b** (assembly phase).



$$\mathbf{A}(i,j) = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, dx \qquad\qquad i,j = 1\ldots,N_h$$





- on the current element $\mathcal{K}$, assemble $\mathbf{A}_\mathcal{K} \in \mathbb{R}^{3\times 3}$

$$\begin{aligned}\mathbf{A}_\mathcal{K}(i,j) &= \int_\mathcal{K} \nabla\varphi_j \cdot \nabla\varphi_i \, dx \\ &= \det(\mathbf{B}_\mathcal{K}) \underbrace{\int_{\widehat{\mathcal{K}}} (\mathbf{B}_\mathcal{K}^{-T}\widehat{\nabla}\widehat{\varphi}_j)\cdot(\mathbf{B}_\mathcal{K}^{-T}\widehat{\nabla}\widehat{\varphi}_i)\, d\hat{x}}_{\text{quadrature formulas}} \quad i,j = 1\ldots,3\end{aligned}$$

In fact, (if gradients are columns) by chain rule we have

$$\widehat{\nabla}\widehat{\varphi}_j = \mathbf{B}_{\mathcal{K}}^T \nabla\varphi_j \quad \text{or} \quad \nabla\varphi_j = \mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla}\widehat{\varphi}_j$$

Then,

$$\nabla\varphi_j \cdot \nabla\varphi_i = \nabla\varphi_j^T \nabla\varphi_i = \widehat{\nabla}\widehat{\varphi}_j^T \mathbf{B}_{\mathcal{K}}^{-1} \mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla}\widehat{\varphi}_i$$

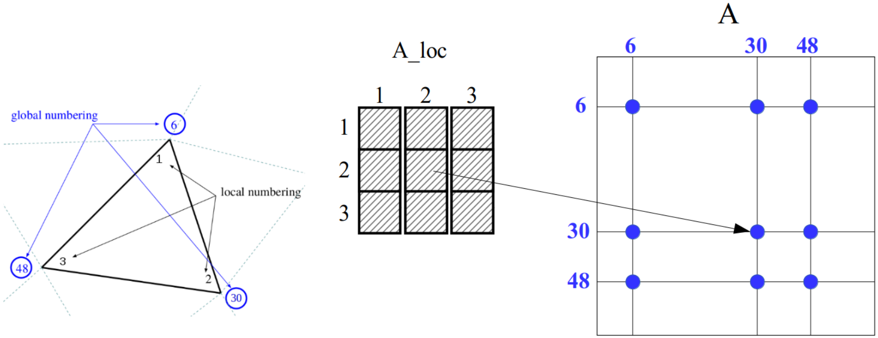
On the other hand, $\mathbf{B}_{\mathcal{K}}$ is constant, and thus

$$\begin{aligned} \mathbf{A}_{\mathcal{K}}(i,j) = \int_{\mathcal{K}} \nabla\varphi_j \cdot \nabla\varphi_i \, dx &= \det(\mathbf{B}_{\mathcal{K}}) \int_{\hat{\mathcal{K}}} \widehat{\nabla}\widehat{\varphi}_j^T \mathbf{B}_{\mathcal{K}}^{-1} \mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla}\widehat{\varphi}_i \, d\hat{x} \\ &= \frac{\det(\mathbf{B}_{\mathcal{K}})}{2} \widehat{\nabla}\widehat{\varphi}_j^T \mathbf{B}_{\mathcal{K}}^{-1} \mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla}\widehat{\varphi}_i \end{aligned}$$

2. Assembly of the stiffness matrix A

Idea: Loop over the elements $\mathcal{K}$ and for each element do:

- compute all the integrals on the element (local matrix **A_loc**);
- add the computed integrals at the proper positions of the stiffness matrix **A** (assembly phase).



A quick look into the code: the **basis** structure

$\boxed{\text{basis} = \text{C\_shape\_basis(Dati)}}$

Definition of shape functions $\widehat{\varphi}$ and their gradients $\widehat{\nabla\varphi}$ is given by the `basis` structure :

```
basis =

  1x3 struct array with fields:

    num
    n_edge
    fbases
    Gbases_1
    Gbases_2
```

num	n_edge	fbases $\hat{\varphi}$	Gbases_1 $\frac{\partial \hat{\varphi}}{\partial \hat{x}}$	Gbases_2 $\frac{\partial \hat{\varphi}}{\partial \hat{y}}$
3	3	'-1.*csi - 1.*eta + 1'	'-1 + 0.*eta + 0'	'0 .*csi - 1 + 0'
3	3	' 1.*csi + 0.*eta + 0'	' 1 + 0.*eta + 0'	'0 .*csi + 0 + 0'
3	3	' 0.*csi + 1.*eta + 0'	' 0 + 0.*eta + 0'	'0 .*csi + 1 + 0'

Compute integrals through quadrature formulas

$$\boxed{\text{[nodes\_2D, w\_2D] = C\_quadrature(Dati);}}$$

To integrate $\int_{\mathcal{K}} g \, dx$ for a generic function g we use the quadrature rule

$$\int_{\mathcal{K}} g \, dx \approx \sum_{q=1}^{\text{nqn}} g(\mathbf{x}_q) w_q \det(\mathbf{B}_{\mathcal{K}})$$

where $\mathbf{x}_q$ are suitable quadrature points, w_q are the associated weights and $\det(\mathbf{B}_{\mathcal{K}})$ is the determinant of the jacobian of the transformation $\mathbf{F}_{\mathcal{K}}$)

- $\mathbf{x}_q \rightarrow$ `nodes_2D`
- $w_q \rightarrow$ `w_2D`

Compute integrals through quadrature formulas

$$\boxed{[\text{nodes\_2D, w\_2D}] = \text{C\_quadrature(Dati);}}$$

It is possible to set different quadrature rules by changing the values of **nqn** (i.e. `Dati.nqn_2D` in `Dati.m`).
Here below a list of possible choices:

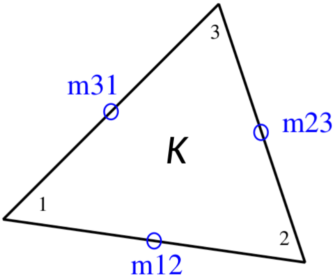
- **nqn**=1, degree of precision: 1
- **nqn**=3, degree of precision: 2
- **nqn**=4, degree of precision: 3
- **nqn**=7, degree of precision: 4
- ... see C_Tria_int_2D.m

Mid-Point quadrature formula

Example: **nqn** $= 3$ leads to the mid-point quadrature formula

$$\int_{\mathcal{K}} g \, dx \approx \frac{|\mathcal{K}|}{3} \left[g(m_{12}) + g(m_{23}) + g(m_{31}) \right]$$

which is exact for quadratic polynomials.



A quick look into the code: the **dphiq** array

$\boxed{\text{[dphiq,Grad] = C\_evalshape(basis,nodes\_2D)}}$

Evaluation of the shape functions $\widehat{\varphi}$ at quadrature nodes (on the reference element). For the mid-point rule we have

```
dphiq(:,:,1) = [0.5000 0 0.5000]
dphiq(:,:,2) = [0.5000 0.5000 0]
dphiq(:,:,3) = [ 0 0.5000 0.5000]
```

$\boxed{\texttt{dphiq(:,q,j)} \leftarrow \widehat{\varphi}_j(\mathbf{x}_q)}$

A quick look into the code: the **Grad** array

$\boxed{\text{[dphiq,Grad]} = \text{C\_evalshape(basis,nodes\_2D)}}$

Evaluation of $\widehat{\nabla}\widehat{\varphi}$ at quadrature nodes (on the reference element). For any quadrature rule we have

```
Grad(:,:,1) = [-1 -1; -1 -1; -1 -1]
Grad(:,:,2) = [1 0; 1 0; 1 0]
Grad(:,:,3) = [0 1; 0 1; 0 1]
```

$\boxed{\texttt{Grad(q,:,j)} \leftarrow \widehat{\nabla}\widehat{\varphi}_j(\mathbf{x}_q)}$

How to implement Dirichlet boundary conditions

Dirichlet boundary conditions can be enforced as follows:

- If $\mathbf{x}_i \in \partial\Omega$ change the i-th equation of the system:
  - set the i-th row of $\mathbf{A}$ to $\mathbf{e}_i^T = (0, ..., 1, ..., 0)$,
  - set the right-hand side $\mathbf{b}_i$ equal to $g(\mathbf{x}_i)$.

  Attention: the matrix $\mathbf{A}$ will loose the symmetry.
- Otherwise:
  - compute the vector $\mathbf{u}_g$ s.t. $\mathbf{u}_g(i) = g(\mathbf{x}_i)$ if $\mathbf{x}_i \in \partial\Omega$, and 0 if $\mathbf{x}_i \notin \partial\Omega$,
  - compute the vector $\mathbf{b}_g = \mathbf{b} - \mathbf{A}\mathbf{u}_g$,
  - set the i-th row of $\mathbf{A}$ to $\mathbf{e}_i^T$ if $\mathbf{x}_i \in \partial\Omega$,
  - set the i-th column of $\mathbf{A}$ to $\mathbf{e}_i$ if $\mathbf{x}_i \in \partial\Omega$,
  - set $\mathbf{b}_g(i) = 0$ if $\mathbf{x}_i \in \partial\Omega$
  - solve the system $\mathbf{A}\mathbf{u} = \mathbf{b}_g$
  - update $\mathbf{u} \longleftarrow \mathbf{u} + \mathbf{u}_g$.

  Note that this is equivalent of introducing the lifting operator.

Code Structure CG_FEM_START

```
CG_FEM_START
├── Assembly                         } 3. Assembly of linear system and right hand side
│   └── C_matrix2D.m
├── BoundaryConditions               } 4. Assignment of Dirichlet boundary conditions
│   └── C_bound_cond2D.m
├── C_dati.m       Data structure → see next slide
├── C_main2D.m     Main program → solution of an elliptic PDE
├── Errors                           } 6. Error analysis
│   ├── C_compute_errors.m
│   └── C_error_L2_H1.m
├── FESpace                          } 2. Finite element space definition
│   ├── C_create_femregion.m
│   ├── C_evalshape.m
│   ├── C_gauleg.m
│   ├── C_get_Jacobian.m
│   ├── C_quadrature.m
│   ├── C_shape_basis.m
│   └── C_Tria_int_2D.m
├── MeshGeneration                   } 1. Mesh generation
│   ├── C_create_mesh.m
│   └── Tria
│       ├── C_create_mesh_tria.m
│       ├── C_jigglemesh.m
│       ├── C_refine_mesh.m
│       └── C_unstructured_mesh.m
└── Postprocessing                   } 5. Visualization of the solution
    ├── C_eval_exact_sol.m
    ├── C_plot_mesh.m
    ├── C_pointwise_sol.m
    └── C_postprocessing.m
```

A quick look into the code: the data structure **Dati**

C_dati.m

```
Dati.name              ->  Test name
Dati.domain            ->  extrema of the rectangular domain [xmin xmax; ymin ymax]
Dati.exact_sol         ->  exact solution (error analysis and Dirichlet conds)
Dati.force             ->  forcing term
Dati.grad_exact_1      ->  gradx of the exact solution
Dati.grad_exact_2      ->  grady of the exact solution
Dati.fem               ->  finite element space ('P1')
Dati.nqn_1D            ->  quadrature rule for line integrals
Dati.nqn_2D            ->  quadrature rule for surface integrals
Dati.MeshType          ->  TS/TU (structured/unstructured)
Dati.refinement_vector ->  refinement levels for error analysis
Dati.visual_graph      ->  graphical visualization of the solution (Y/N)
Dati.plot_errors       ->  compute H1 and L2 errors (Y/N)
```