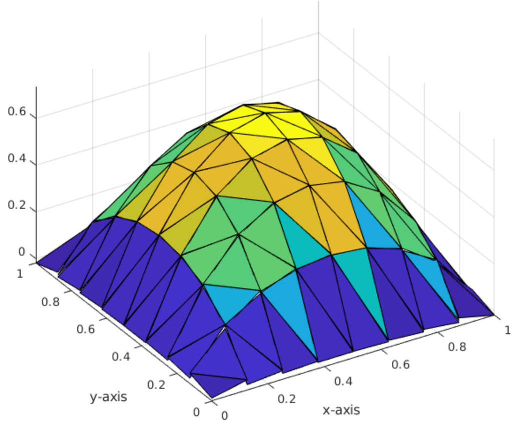
The Discontinuous Galerkin Finite Element Method: Implementation

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

Practical implementation of DG finite elements on a fixed mesh



Poisson's problem:

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

Symmetric Interior Penalty method for the Poisson's problem (g=0)

Variational formulation

$$\text{Find } u_h \in V_h^k \quad \text{s.t.} \quad \mathcal{A}(u_h, v_h) = \int_\Omega f v_h \, dx \quad \forall\, v_h \in V_h^k$$

$$\begin{aligned}\mathcal{A}(w, v) = &\sum_{\mathcal{K} \in \mathcal{T}_h} \int_{\mathcal{K}} \nabla w \cdot \nabla v \, dx - \sum_{F \in \mathcal{F}} \int_F \{\!\{\nabla_h w\}\!\} \cdot [\![v]\!] \, ds \\ &- \sum_{F \in \mathcal{F}} \int_F [\![w]\!] \cdot \{\!\{\nabla_h v\}\!\} \, ds + \sum_{F \in \mathcal{F}} \int_F \gamma \, [\![w]\!] \cdot [\![v]\!] \, ds \quad \forall w, v \in V_h^k.\end{aligned}$$ [1]

[1] We refer the reader to the lecture notes for the definition of the notation used here.

Algebraic formulation

- Fix a basis for V_h^k, i.e.

$$V_h^k = \mathrm{span}\{\varphi_i,\ i = 1\ldots, N_h\},$$

where, $\varphi_i \in L^2(\mathcal{T}_h)$ is s.t. $\varphi_i \in \mathbb{P}^k(\mathcal{K})$ 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 $\mathbf{u} = [u_1, u_2, \ldots, u_{N_h}]^T \in \mathbb{R}^{N_h}$ such that

$$\sum_{j=1}^{N_h} u_j \mathcal{A}(\varphi_j, \varphi_i) = \int_\Omega f \varphi_i \, dx \quad \forall\, i = 1\ldots, N_h$$

Algebraic formulation (cont'd)

Algebraic formulation

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

where

$$\begin{aligned} \mathbf{A}(i,j) &= \mathcal{A}(\varphi_j, \varphi_i) & \qquad i,j = 1\ldots, N_h \\ \mathbf{b}(i) &= \int_\Omega f\varphi_i \, dx & \qquad i = 1\ldots, N_h \end{aligned}$$

Some notation

Stifness matrix

$$\mathbf{A} = \mathbf{V} - \mathbf{I}^T - \mathbf{I} + \mathbf{S}$$

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

$$\mathbf{I}(i,j) = \sum_{F\in\mathcal{F}} \int_F [\![\varphi_j]\!] \cdot \{\!\{\nabla_h\varphi_i\}\!\} \, ds \qquad i,j = 1\ldots,N_h$$

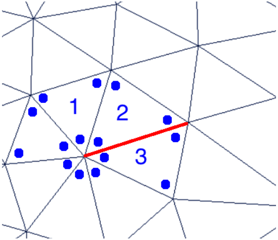
$$\mathbf{S}(i,j) = \sum_{F\in\mathcal{F}} \int_F \gamma \, [\![\varphi_j]\!] \cdot [\![\varphi_i]\!] \, ds \qquad i,j = 1\ldots,N_h$$

Implementation

As for continuous finite elements, 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



[region] = generate_mesh(Data,nRef);

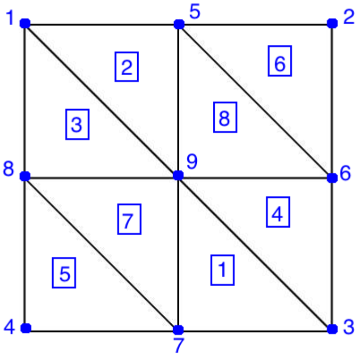
- For any element $\mathcal{K} \in \mathcal{T}_h$ there are $\mathtt{nln} = (k+1)(k+2)/2$ degrees of freedom (•)
- If the degrees of freedom (dof) are numbered elementwise, then the dof associated to the element $\mathcal{K}$ are

$$[\mathtt{nln}(\kappa-1)+1, \mathtt{nln}(\kappa-1)+2, \ldots, \mathtt{nln}(\kappa-1)+\mathtt{nln}]$$

  where κ is the index identifying the element $\mathcal{K}$
- Total number of dof is $N_h = \mathtt{ne} \times \mathtt{nln}$, where $\mathtt{ne}$ is the number of mesh elements.

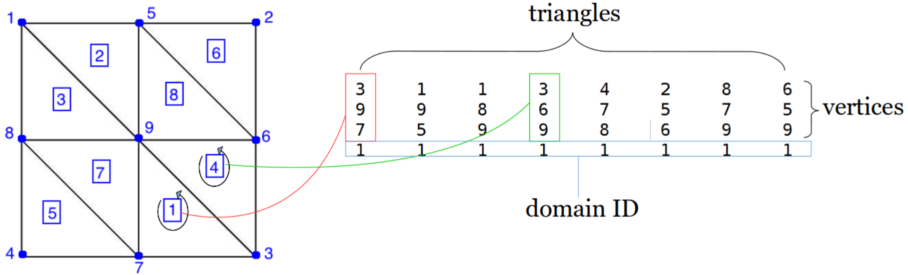
A quick look into the code: the **region** structure

```
region.dim              ->  dimension (2)
femregion.domain        -> (2x2 matrix, real) domain limits
region.h                ->  mesh size
region.coord            -> coordinates of the mesh vertices
region.connectivity     -> connectivity matrix
region.coords_element -> coordinates of dof (repeated nodes!)
region.boundary_edges -> connectivity of boundary edges.
region.degree           -> polynomial degree
```



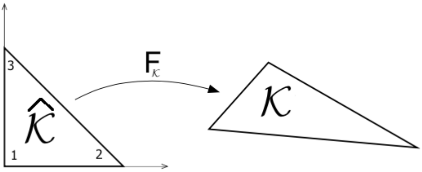
A quick look into the code: the **region** structure

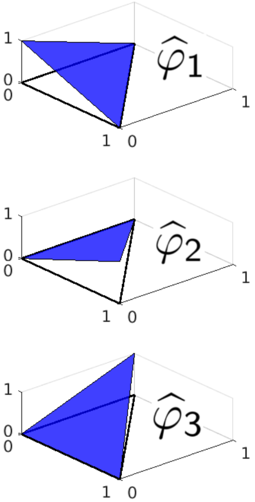
`region.connectivity   -> connectivity of the mesh triangles`



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

Linear shape functions on the reference triangle





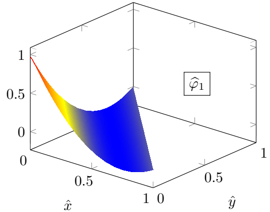
$$
\begin{cases}
\widehat{\varphi}_1(\xi,\eta) = 1-\xi-\eta & \text{node } (0,0) \to (x_1,y_1),\\
\widehat{\varphi}_2(\xi,\eta) = \xi & \text{node } (1,0) \to (x_2,y_2),\\
\widehat{\varphi}_3(\xi,\eta) = \eta & \text{node } (0,1) \to (x_3,y_3),
\end{cases}
$$

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

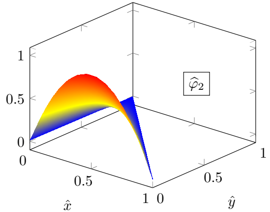
$$\boxed{\text{Then } \varphi_{j|_{\mathcal{K}}} = \widehat{\varphi}_j \circ \mathbf{F}_{\mathcal{K}}^{-1} \text{ and } \widehat{\varphi}_j = \varphi_j \circ \mathbf{F}_{\mathcal{K}}.}$$

Quadratic shape functions on the reference triangle

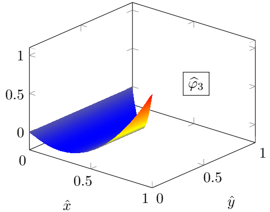
$\widehat{\varphi}_1 = 2\xi^2 + 2\eta^2 + 4\xi\eta - 3\xi - 3\eta + 1$



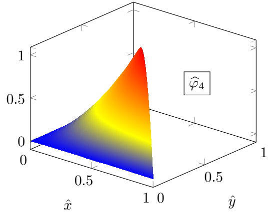
$\widehat{\varphi}_2 = -4\xi^2 - 4\xi\eta + 4\xi$



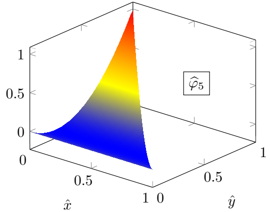
$\widehat{\varphi}_3 = 2\xi^2 - \xi$



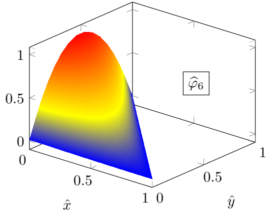
$\widehat{\varphi}_4 = 4\xi\eta$



$\widehat{\varphi}_5 = 2\eta^2 - \eta$



$\widehat{\varphi}_6 = -4\eta^2 - 4\xi\eta + 4\eta$



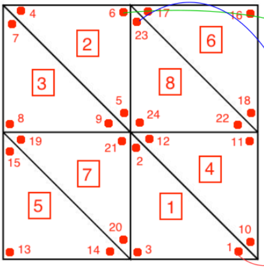
A quick look into the code: the **femregion** struct

$\boxed{[\text{femregion}] = \text{create\_dof}(\text{Data},\text{Region})}$

```
femregion.fem              -> 'P1' or 'P2' or 'P3'
femregion.domain           -> Domain bounds
femregion.h                -> mesh size
femregion.nedges           -> number of element edges
femregion.nln              -> number of local dof
femregion.ndof             -> number of global dof
femregion.ne               -> number of mesh elements
femregion.dof              -> coordinates of global dof
femregion.nqn              -> number of 1D quadrature nodes
femregion.degree           -> polynomial degree
femregion.connectivity     -> connectivity matrix
femregion.coords_element -> duplication of femregion.dof
femregion.boundary_edges -> connectivity of boundary edges
```

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

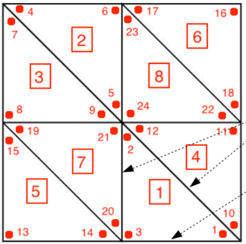
```
femregion.dof   -> list of dof (repeated entries)
```



duplicate vertices

x	y
1.0000	0
0.5000	0.5000
0.5000	0
0	1.0000
0.5000	0.5000
0.5000	1.0000
0	1.0000
0	0.5000
0.5000	0.5000
1.0000	0
1.0000	0.5000
0.5000	0.5000
0	0
0.5000	0
0	0.5000
1.0000	1.0000
0.5000	1.0000
1.0000	0.5000
0	0.5000
0.5000	0
0.5000	0.5000
1.0000	0.5000
0.5000	1.0000
0.5000	0.5000

$$\boxed{[\text{neighbour}] = \text{neighbours}(\text{femregion})}$$



neigh			neighedges		
4	7	-1	3	2	-1
3	8	-1	3	2	-1
-1	7	2	-1	3	1
-1	8	1	-1	3	1
-1	7	-1	-1	1	-1
-1	8	-1	-1	1	-1
5	1	3	2	2	2
6	2	4	2	2	2

edges

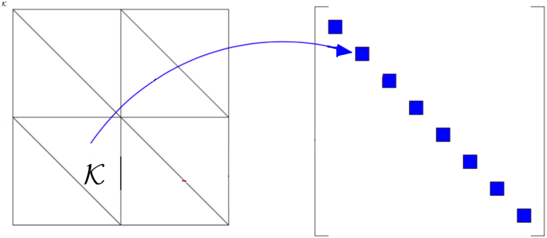
neighboring elements

neighboring edges

Element **1** (edge **1**) has neighbor element **4** (edge **3**)
Element **1** (edge **2**) has neighbor element **7** (edge **2**)
Element **1** (edge **3**) is a boundary edge (edge -**1**)

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

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



- on the current mesh element $\mathcal{K}$, assemble $\mathbf{V}_\mathcal{K} \in \mathbb{R}^{\mathrm{nln}\times\mathrm{nln}}$

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

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

```
% loop over elements
for ie = 1:femregion.ne
    ...
    % index      -> Local dof to global dof map
    % BJ         -> Jacobian of the elemental map
    % BJinv      -> Inverse Jacobian of the elemental map
    % pphys_2D  -> vertex coordinates in the physical domain
   ...
    % loop over 2D quadrature nodes
    for k = 1:length(w_2D)
        % dx     -> scaled weight for the quadrature formula
        dx = w_2D(k)*det(BJ);
        % assembly of V
        for i = 1 : femregion.nln
            for j = 1 : femregion.nln
                V(index(i),index(j)) = V(index(i),index(j)) ...
                    + ((Grad(k,:,i) * BJinv) * (Grad(k,:,j) * BJinv )') .*dx ;
            end
        end
    end
    ...
end
```

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

$$\mathbf{S}(i,j) = \sum_{F \in \mathcal{F}} \int_F \gamma \, [\![\varphi_j]\!] \cdot [\![\varphi_i]\!] \, ds \qquad\qquad i,j = 1\ldots, N_h$$

- Let $\mathcal{K}^+$ be the current element, and let $\mathcal{K}^-$ be its neighbour through the interior edge F.
- Using the definition of the jump operator we have

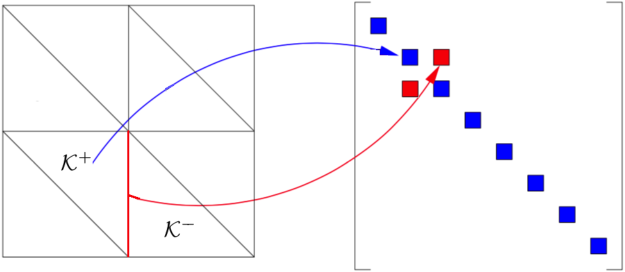
$$\begin{aligned}
\int_F \gamma \, [\![\varphi_j]\!] \cdot [\![\varphi_i]\!] \, ds &= \int_F \gamma(\varphi_j^+ \mathbf{n}^+ + \varphi_j^- \mathbf{n}^-) \cdot (\varphi_i^+ \mathbf{n}^+ + \varphi_i^- \mathbf{n}^-) \, ds \\
&= \underbrace{\int_F \gamma \varphi_i^+ (\varphi_j^+ - \varphi_j^-) \, ds}_{\substack{\text{Assemble this since } \mathcal{K}^+ \\ \text{is the current element}}} + \underbrace{\int_F \gamma \varphi_i^- (\varphi_j^- - \varphi_j^+) \, ds}_{\substack{\text{Assemble this when } \mathcal{K}^- \\ \text{will be the current element}}}
\end{aligned}$$

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

- On each interior edge F of the element $\mathcal{K}^+$ assemble

$$\mathbf{S}_F^D \in \mathbb{R}^{\mathsf{nln}\times\mathsf{nln}} \qquad \mathbf{S}_F^D(i,j) = \int_F \gamma \varphi_i^+ \varphi_j^+ \, ds = |F| \underbrace{\int_{\widehat{F}} \gamma \widehat{\varphi}_i^+ \widehat{\varphi}_j^+ \, ds}_{quadrature}$$

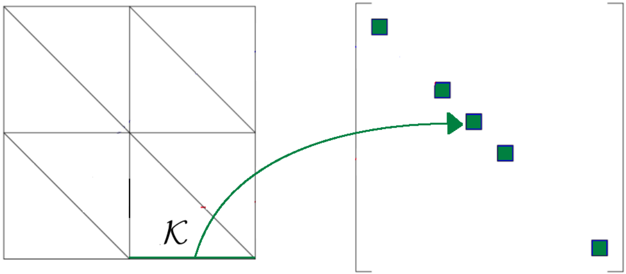
$$\mathbf{S}_F^N \in \mathbb{R}^{\mathsf{nln}\times\mathsf{nln}} \qquad \mathbf{S}_F^N(i,j) = -\int_F \gamma \varphi_i^+ \varphi_j^- \, ds = -|F| \underbrace{\int_{\widehat{F}} \gamma \widehat{\varphi}_i^+ \widehat{\varphi}_j^- \, ds}_{quadrature}$$



2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

- On each boundary edge F of the element $\mathcal{K}$ assemble

$$\mathbf{S}_F^D \in \mathbb{R}^{\mathrm{nln}\times\mathrm{nln}} \qquad \mathbf{S}_F^D(i,j) = \int_F \gamma \varphi_i^+ \varphi_j^+ \, ds = |F| \underbrace{\int_{\widehat{F}} \gamma \widehat{\varphi}_i^+ \widehat{\varphi}_j^+ \, ds}_{quadrature}$$



2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

```matlab
% loop over elements
for ie = 1:femregion.ne
    ...
    for iedg = 1 : neighbour.nedges  % Loop over the triangle's  edges
        ...
        % penalty_scaled -> penalty term
        for k = 1:nqn_1D   % loop over 1D quadrature nodes
            for i = 1:femregion.nln
                for j = 1 : femregion.nln
                    % ds  -> scaled weight for the quadrature formula

                    % B_edge(i,k,iedg) -> phi_i^+(x_k) on the iedg-th edge
                    S(index(i),index(j)) = S(index(i),index(j)) + penalty_scaled ...
                                               .* B_edge(i,k,iedg) .* B_edge(j,k,iedg) .* ds;

                    % B_edge(i,kk,neigedge) -> phi_i^-(x_kk) on the neighedge-th edge
                        SN(i,j,iedg) = SN(i,j,iedg) ...
                                           - penalty_scaled .* B_edge(i,k,iedg) ...
                                                          .* B_edge(j,kk,neigedge) .* ds;
                    end
                end

            end
        end
    end
    % assembly of S -> sum of block diagonal S with extra diagonal entries SN
    [S] = assemble_neigh(S,index,neigh_ie,SN,femregion.nln,neighbour.nedges);
end
```

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

$$\mathbf{I}(i,j) = \sum_{F\in\mathcal{F}} \int_F \{\!\{\nabla_h \varphi_i\}\!\} \cdot [\![\varphi_j]\!] \, ds \qquad\qquad i,j = 1\ldots,N_h$$

- Let $\mathcal{K}^+$ be the current element and let $\mathcal{K}^-$ be its neighbor through the interior edge F.
- Using the definition of the jump and average operators we have

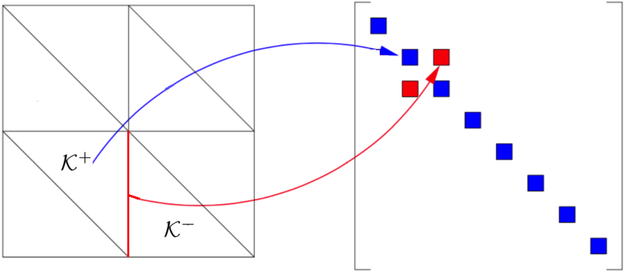
$$\begin{aligned}\int_F \{\!\{\nabla_h \varphi_i\}\!\} \cdot [\![\varphi_j]\!] \, ds &= \int_F \frac{1}{2}(\nabla_h \varphi_i^+ + \nabla_h \varphi_i^-) \cdot (\varphi_j^+ \mathbf{n}^+ + \varphi_j^- \mathbf{n}^-) \, ds \\ &= \underbrace{\frac{1}{2}\int_F \nabla_h \varphi_i^+ \cdot \mathbf{n}^+ (\varphi_j^+ - \varphi_j^-) \, ds}_{\substack{\text{Assemble this since } \mathcal{K}^+ \\ \text{is the current element}}} \\ &\quad + \underbrace{\frac{1}{2}\int_F \nabla_h \varphi_i^- \cdot \mathbf{n}^- (\varphi_j^- - \varphi_j^+) \, ds}_{\substack{\text{Assemble this when } \mathcal{K}^- \\ \text{will be the current element}}}\end{aligned}$$

2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

- On each interior edge F of the element $\mathcal{K}^+$ assemble

$$\mathbf{I}_F^D \in \mathbb{R}^{\text{nln}\times\text{nln}} \quad \mathbf{I}_F^D(i,j) = \frac{1}{2}\int_F \nabla_h \varphi_i^+ \cdot \mathbf{n}^+ \varphi_j^+ \, ds = \frac{|F|}{2} \underbrace{\int_{\widehat{F}} (\mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla} \widehat{\varphi}_i^+) \cdot \mathbf{n}^+ \widehat{\varphi}_j^+ \, ds}_{quadrature}$$

$$\mathbf{I}_F^N \in \mathbb{R}^{\text{nln}\times\text{nln}} \quad \mathbf{I}_F^N(i,j) = -\frac{1}{2}\int_F \nabla_h \varphi_i^+ \cdot \mathbf{n}^+ \varphi_j^- \, ds = -\frac{|F|}{2} \underbrace{\int_{\widehat{F}} (\mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla} \widehat{\varphi}_i^+) \cdot \mathbf{n}^+ \widehat{\varphi}_j^- \, ds}_{quadrature}$$

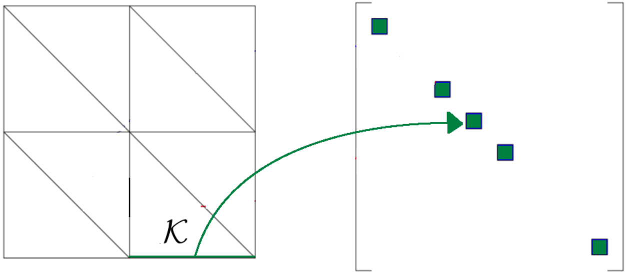


2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

- On each boundary edge F of the element $\mathcal{K}$ assemble

$$\mathbf{I}_F^D \in \mathbb{R}^{\mathrm{nln}\times\mathrm{nln}} \qquad \mathbf{I}_F^D(i,j) = \int_F \nabla_h \varphi_i^+ \cdot \mathbf{n}^+ \varphi_j^+ \, ds$$

$$= |F| \underbrace{\int_{\widehat{F}} (\mathbf{B}_{\mathcal{K}}^{-T} \widehat{\nabla} \widehat{\varphi}_i^+) \cdot \mathbf{n}^+ \widehat{\varphi}_j^+ \, ds}_{quadrature}$$



2. Assembly of the stiffness matrix $\mathbf{A} = V - I^T - I + S$

```matlab
% loop over elements
for ie = 1:femregion.ne
    ...
    for iedg = 1 : neighbour.nedges  % Loop over the triangle's  edges
        ...
        for k = 1:nqn_1D   % loop over 1D quadrature nodes
           for i = 1:femregion.nln
              for j = 1 : femregion.nln
                % B_edge(i,k,iedg)           -> phi_i^+(x_k) on the iedg-th edge
                % G_edge(k,:,i,iedg)*BJinv   -> grad(phi_i^+(x_k) on the iedg-th edge
                % normals(:,iedg)            -> normal vector for the iedg-th edge
                if neigh_ie(iedg) ~= -1    % Internal faces
                      I(index(i),index(j)) = I(index(i),index(j)) ...
                          + 0.5 .* ((G_edge(k,:,i,iedg)*BJinv)*normals(:,iedg)) ...
                                .* B_edge(j,k,iedg) .* ds;
                % B_edge(i,kk,neigedge) -> phi_i^-(x_kk) on the neighedge-th edge
                      IN(i,j,iedg) = IN(i,j,iedg) - 0.5 .* B_edge(j,kk,neigedge) ...
                              .* ((G_edge(k,:,i,iedg)*BJinv)*normals(:,iedg)) .* ds;
                elseif neigh_ie(iedg) == -1  % Boundary faces
                      I(index(i),index(j))  = I(index(i),index(j)) ...
                              +  ((G_edge(k,:,i,iedg)*BJinv)*normals(:,iedg)) ...
                                 .* B_edge(j,k,iedg) .* ds;
                end
              end
           end
        end
    end
 end
   % assembly of I -> sum of block diagonal I with extra diagonal entries IN
    [I] = assemble_neigh(I,index,neigh_ie,IN,femregion.nln,neighbour.nedges);
end
```

2. Assembly of the right-hand side

$$\mathbf{b}(i) = F(\varphi_i) = \int_\Omega f \varphi_i \, dx \qquad\qquad i = 1 \ldots, N_h$$

On the current element $\mathcal{K}$, assemble $\mathbf{b}_\mathcal{K} \in \mathbb{R}^{\mathrm{nln}}$

$$\mathbf{b}_\mathcal{K}(i) = \underbrace{\int_\mathcal{K} f \varphi_i \, dx}_{\text{quadrature}} \qquad\qquad i = 1 \ldots, \mathrm{nln}$$

How to include non-homogeneous Dirichlet boundary conditions?

How to implement Dirichlet boundary conditions

Dirichlet boundary conditions can be enforced by penalization. For each boundadry edge F of the element $\mathcal{K}$ we consider the following definition

$$[\![u]\!] = (u - g)\mathbf{n}.$$

This leads to the following modification of the forcing term

$$\mathbf{b}_{\mathcal{K}}(i) = \int_{\mathcal{K}} f\varphi_i \, dx - \int_F g\nabla_h \varphi_i^+ \cdot \mathbf{n} \, ds + \int_F \gamma g \varphi_i^+ \, ds \quad i = 1 \ldots, \mathrm{nln}$$

How to implement Dirichlet boundary conditions

```
for ie = 1:femregion.ne
    ...
    for k = 1:length(w_2D) % loop over 2D quadrature nodes
        for i = 1 : femregion.nln
           % F     -> val of rhs on quad nodes
           % dphiq -> val of shape function phi_i^+(x_k) on quad nodes
           f(index(i)) = f(index(i)) + F*dphiq(1,k,i).*dx;
        end
    end
    for iedg = 1 : neighbour.nedges  % Loop over the triangle's  edges
        ...
        for k = 1:nqn_1D   % loop over 1D quadrature nodes
          for i = 1:femregion.nln % loop over local dof

            if  neigh_ie(iedg) == -1  % Boundary face
              ...
              % gd = val of Dirichlet boundary conditions on quad nodes
              f(index(i)) = f(index(i))+penalty_scaled.*B_edge(i,k,iedg).*gd.*ds ;
              f(index(i)) = f(index(i))- gd.*ds ...
                                 .* ((G_edge(k,:,i,iedg)*BJinv)*normals(:,iedg));
            end
        end
    end
end
```

Code Structure DG_FEM

```
DG_FEM/
├── Assembly
│   ├── assemble_neigh.m
│   └── matrix2D.m
├── convergence_test.m
├── dati.m
├── Errors
│   └── compute_errors.m
├── FESpace
│   ├── basis_lagrange.m
│   ├── create_dof.m
│   ├── evalshape.m
│   ├── get_jacobian_physical_points.m
│   ├── get_normals_meshsize_faces.m
│   ├── get_physical_points_faces.m
│   ├── matrix_coeff_P1.m
│   ├── matrix_coeff_P2.m
│   ├── matrix_coeff_P3.m
│   ├── quadrature.m
│   └── swap_vector.m
├── main2D.m
├── MeshGeneration
│   ├── generate_mesh.m
│   ├── jiggle_mesh.m
│   ├── neighbours.m
│   └── refine_mesh.m
└── PostProcessing
    ├── plot_mesh.m
    └── postprocessing.m
```

- Assembly: 3. Assembly of linear system and right hand side
- convergence_test.m: Error analysis → verification of the numerical method
- dati.m: Data structure → see next slide
- Errors: 5. Error analysis
- FESpace: 2. Finite element space definition
- main2D.m: Main program → solution of an elliptic PDE
- MeshGeneration: 1. Mesh generation
- PostProcessing: 4. Visualization of the solution

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

dati.m

```
function [DATA] = dati(test)

if test=='Test1'

DATA = struct(  'name',               test,...
                ... % Test name
                 'method',             'SIP',...
                ... % Set DG discretization
                'domain',             [0,1;0,1],...
                ... % Domain bounds
                'exact_sol',          '...',...
                ... % Definition of exact solution
                'source',             '...',...
                ... % Forcing term
                'grad_exact_1',       '...',...
                ... % Definition of exact gradient (x comp)
                'grad_exact_2',       '...',...
                ... % Definition of exact gradient (y comp)
                'fem',                'P1',...
                ... % Finite element space (other choices 'P2', 'P3')'
                'penalty_coeff',       10,...
                ... % Penalty coefficient
                'nqn',                 4 ...
                ... % Number of 1d GL quadrature nodes
                );
end
```