Elementi Finiti - Esercitazione 4
Assemblaggio locale e globale

Prof. Giancarlo Sangalli   Ivan Bioli

02 Aprile 2025

1 Ripasso sull'assemblaggio del Metodo agli Elementi Finiti

Dato una forma bilineare $a : \mathrm{V} \times \mathrm{V} \to \mathbb{R}$, continua e coerciva, e un funzionale $f \in \mathrm{V}'$, consideriamo il problema variazionale associato:

$$\text{Trovare } u \in \mathrm{V} \text{ tale che } a(u, v) = \langle f,\, v \rangle_{\mathrm{V}' \times \mathrm{V}}\,, \qquad \forall v \in \mathrm{V}. \tag{1}$$

Inizialmente, consideriamo il caso con condizioni al bordo di Dirichlet omogenee, ovvero $\mathrm{V} = \mathrm{H}^1_0(\Omega)$. In seguito, analizzeremo il caso di condizioni al bordo diverse.

Per discretizzare il problema (1), introduciamo lo spazio finito-dimensionale

$$\mathrm{V}_h = \mathbb{P}^1(\mathcal{T}_h) \cap \mathrm{H}^1_0(\Omega) \subset \mathrm{V},$$

e approssimiamo a e f con $a_h \approx a$ e $f_h \approx f$, tenendo conto dell'uso della quadratura numerica. Il problema variazionale discreto risulta quindi:

$$\text{Trovare } u_h \in \mathrm{V}_h \text{ tale che } a_h(u_h, v_h) = \langle f_h,\, v_h \rangle_{\mathrm{V}_h' \times \mathrm{V}_h}\,, \qquad \forall v_h \in \mathrm{V}_h. \tag{2}$$

Poiché V_h è uno spazio finito-dimensionale, possiamo considerare una base $\{\varphi_i\}_{i=1}^N$ costituita dalle funzioni di forma (hat-functions) associate ai nodi *interni* della mesh. Espandendo u_h nella base:

$$u_h(x) = \sum_{j=1}^{N} u_j \varphi_j(x),$$

il problema (2) si traduce nel sistema algebrico per il vettore incognito $\mathbf{u} = [u_1, \ldots, u_N]$:

$$\text{Trovare } \mathbf{u} \in \mathbb{R}^N \text{ tale che } \sum_{j=1}^{N} u_j \underbrace{a_h(\varphi_j,\, \varphi_i)}_{[\mathbf{A}]_{ij}} = \underbrace{\langle f_h,\, \varphi_i \rangle_{\mathrm{V}_h' \times \mathrm{V}_h}}_{[\mathbf{f}]_i}, \qquad \forall i = 1, \ldots, N.$$

In forma matriciale, questo sistema può essere scritto come:

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \qquad [\mathbf{A}]_{ij} = a_h(\varphi_j,\, \varphi_i), \qquad [\mathbf{f}]_i = \langle f_h,\, \varphi_i \rangle_{\mathrm{V}_h' \times \mathrm{V}_h}\,. \tag{3}$$

$\mathbf{A}$ è la *stiffness matrix* e $\mathbf{f}$ il *load vector*.

Per assemblare il sistema (3), consideriamo come esempio il caso di

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x}.$$

Un algoritmo semplice, ma computazionalmente costoso, per costruire la matrice $\mathbf{A}$ è il seguente:

$\mathbf{A} = \mathbf{0}$
for $i = 1 : N$ **do**
  for $j = 1 : N$ **do**
    for $T \in \mathcal{T}_h$ **do**
      $[\mathbf{A}]_{ij} = [\mathbf{A}]_{ij} + \int_T \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}\mathbf{x}$
    end for
  end for
end for

Tuttavia, si osserva che l'integrale $\int_T \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}\mathbf{x}$ è diverso da zero solo se φ_j e φ_i sono funzioni di forma associate ai vertici di T. Pertanto, un metodo più efficiente consiste nel calcolare gli integrali $\int_T \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}\mathbf{x}$ per ogni elemento T della mesh, considerando solo le funzioni di forma corrispondenti ai vertici di T, e successivamente sommare i contributi nelle posizioni appropriate della matrice $\mathbf{A}$.

Sia $T_k \in \mathcal{T}_h$ il k-esimo triangolo della mesh. Denotiamo con $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ i suoi vertici e siano rispettivamente $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ i loro indici globali, ovvero $\mathcal{N}_i = \mathtt{T}[i,k]$. Questo ci permette di definire una corrispondenza *local-to-global* $i \mapsto \mathcal{N}_i$ per il triangolo T_k. Possiamo quindi assemblare la *matrice locale* $\mathbf{A}_k \in \mathbb{R}^{3\times 3}$:

$$[\mathbf{A}_k]_{ij} = \int_{T_k} \nabla\varphi_{\mathcal{N}_j} \cdot \nabla\varphi_{\mathcal{N}_i} \mathrm{d}\mathbf{x}$$

e successivamente aggiungere il contributo alla matrice globale come:

$$[\mathbf{A}]_{\mathcal{N}_i\mathcal{N}_j} \leftarrow [\mathbf{A}]_{\mathcal{N}_i\mathcal{N}_j} + [\mathbf{A}_k]_{ij}.$$

Il processo di assemblaggio può essere descritto dal seguente pseudocodice:

$\mathbf{A} = \mathbf{0}$
for $T_k \in \mathcal{T}_h$ **do**
  for $i = 1 : 3$ **do**
    for $j = 1 : 3$ **do**
      $\mathcal{N}_i = \mathtt{T}[i,k]$
      $\mathcal{N}_j = \mathtt{T}[j,k]$
      $[\mathbf{A}]_{\mathcal{N}_i\mathcal{N}_j} \leftarrow [\mathbf{A}]_{\mathcal{N}_i\mathcal{N}_j} + \int_{T_k} \nabla\varphi_{\mathcal{N}_j} \cdot \nabla\varphi_{\mathcal{N}_i} \mathrm{d}\mathbf{x}$
    end for
  end for
end for

In questo caso, gli integrali $\int_{T_k} \nabla\varphi_{\mathcal{N}_j} \cdot \nabla\varphi_{\mathcal{N}_i} \mathrm{d}\mathbf{x}$ possono essere calcolati esattamente. Tuttavia, per forme bilineari più generali, è necessario ricorrere a regole di quadratura numerica.

L'assemblaggio del termine noto segue un procedimento analogo. Se, ad esempio, $\langle f, v\rangle_{\mathrm{V}'\times\mathrm{V}} = \int_\Omega fv$ con $f \in \mathrm{L}^2(\Omega)$, il termine noto viene calcolato come segue:

$\mathbf{f} = \mathbf{0}$
for $T_k \in \mathcal{T}_h$ **do**

for $i = 1 : 3$ **do**
  $\mathcal{N}_i = \mathtt{T}[i, k]$
  $[\mathbf{f}]_{\mathcal{N}_i} \leftarrow [\mathbf{f}]_{\mathcal{N}_i} + \int_{T_k} f \varphi_{\mathcal{N}_i} \mathrm{d}\mathbf{x}$
end for
end for

dove l'integrale è approssimato mediante una formula di quadratura.

2 Assemblaggio locale per il problema di Poisson

Il primo passo per costruire il sistema algebrico (3) è l'assemblaggio delle matrici e dei vettori locali. Useremo formule di quadratura anche per il problema di Poisson, nonostante non sia strettamente necessario per l'assemblaggio delle matrici di stiffness locali.

Esercizio 1

Implementare la funzione

```
@memoize function shapef_2DLFE(quadrule::TriQuad)
```

che prende in input una regola di quadratura `quadrule` di tipo `TriQuad` e valuta le funzioni di base nelle coordinate dei punti di quadratura (sull'elemento di riferimento). Se `quadrule.points` è una matrice $2 \times q$, la funzione deve restituire una matrice $3 \times q$, dove ogni colonna contiene le valutazioni delle tre funzioni di base nel corrispondente punto di quadratura. Potete trovare un codice da completare in `Assembly.jl`. Si noti l'utilizzo della macro di Julia `@memoize` che permette fare caching e non ricalcolare ogni volta la funzione, se viene usata sempre la stessa regola di quadratura. Per saperne di più, vedete la documentazione del pacchetto `Memoize.jl`.

Soluzione dell'esercizio 1

La soluzione si trova nel file `Assembly.jl`.

Esercizio 2

Calcolare i gradienti delle funzioni di base sull'elemento di riferimento.

Soluzione dell'esercizio 2

Le funzioni di base sull'elemento di riferimento sono

$$\varphi_1(x, y) = 1 - x - y, \qquad \varphi_2(x, y) = x, \qquad \varphi_3(x, y) = y.$$

Dunque i gradienti sono

$$\nabla\varphi_1(x, y) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \qquad \nabla\varphi_2(x, y) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \nabla\varphi_3(x, y) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Esercizio 3

Implementare la funzione

```
@memoize function ∇shapef_2DLFE(quadrule::TriQuad)
```

che prende in input una regola di quadratura `quadrule` di tipo `TriQuad` e calcola i gradienti delle funzioni di base nelle coordinate dei punti di quadratura (sull'elemento di riferimento). Se `quadrule.points` è una matrice $2 \times q$, la funzione deve restituire un array matrice $2 \times 3 \times q$. Potete trovare un codice da completare in `Assembly.jl`.

Suggerimento: si consiglia di utilizzare la funzione `repeat` per replicare la matrice dei gradienti (costante) lungo la terza dimensione.

Soluzione dell'esercizio 3

La soluzione si trova nel file `Assembly.jl`.

Esercizio 4

Dimostrare che se l'elemento T_k è ottenuto come immagine dell'elemento di riferimento $\hat{T}$ tramite la mappa affine $F(\hat{\mathbf{x}}) = \mathbf{B}_k\hat{\mathbf{x}} + \mathbf{a}_k$, allora

$$[\mathbf{A}_k]_{ij} = \int_{T_k} \nabla\varphi_{\mathcal{N}_j} \cdot \nabla\varphi_{\mathcal{N}_i} \mathrm{d}\mathbf{x} = \int_{\hat{T}} \mathbf{B}_k^{-\top} \nabla\hat{\varphi}_j \cdot \mathbf{B}_k^{-\top} \nabla\hat{\varphi}_i |\det \mathbf{B}_k| \mathrm{d}\hat{\mathbf{x}} \tag{4}$$

Soluzione dell'esercizio 4

É sufficiente usare la formula di cambio di variabile.

Esercizio 5

Modificare la struttura dati per la mesh `Mesh` aggiungendo un campo `invBk` per memorizzare le inverse delle matrici $\mathbf{B}_k$ che descrivono le trasformazioni affini.

Soluzione dell'esercizio 5

La soluzione si trova nel file `Meshing.jl`.

Esercizio 6

Implementare la funzione

```julia
function get_invBk!(mesh::Mesh)
```

che calcola, restituisce e memorizza le matrici inverse $\mathbf{B}_k^{-1}$ per ogni elemento della mesh. L'array risultante deve avere dimensione $2 \times 2 \times N_{\mathrm{tri}}$. Potete trovare un codice da completare in `Meshing.jl`.

Soluzione dell'esercizio 6

La soluzione si trova nel file `Meshing.jl`.

Esercizio 7

Implementare la funzione

```julia
function poisson_assemble_local!(Ke::Matrix, fe::Vector, mesh::Mesh,
                                 cell_index::Integer, f)
```

che assembla la matrice di stiffness locale e il load vector locale per il problema di Poisson. La funzione prende in input:

- `Ke`: la matrice di stiffness locale da assemblare (non necessariamente già inizializzata a **0**);
- `fe`: il load vector locale da assemblare (non necessariamente già inizializzato a **0**);
- `mesh`: l'oggetto mesh contenente la discretizzazione del dominio;
- `cell_index`: l'indice della elemento corrente corrente;
- `f`: la funzione del termine sorgente;

e restituisce:

- `Ke`: la matrice di stiffness locale assemblata.
- `fe`: i load vector locale assemblato.

Potete trovare un codice da completare in `Assembly.jl`.

Suggerimento 1: usare tutte le funzioni implementate durante questa esercitazione.

Suggerimento 2: usare la regola di quadratura Q_0 per assemblare la stiffness matrix e la regola di quadratura Q_2 per assemblare il load vector.

Soluzione dell'esercizio 7

La soluzione si trova nel file `Assembly.jl`.

3 Da assemblaggio locale ad assemblaggio globale

Esercizio 8

Implementare la funzione

```
function assemble_global(mesh::Mesh, local_assembler!)
```

che, data una mesh e un assembler locale, costruisce la matrice di stiffness globale e il vettore dei termini noti. La funzione accetta come parametri:

- `mesh`: un oggetto contenente la discretizzazione del dominio;
- `local_assembler!`: una funzione che costruisce la matrice di stiffness e il vettore dei termini noti a livello locale, con la seguente sintassi: `local_assembler!(Ke, fe, mesh, cell_index)`.

La funzione restituisce la matrice di stiffness globale $\mathbf{A}$ e il vettore dei termini noti $\mathbf{f}$. L'implementazione deve iterare su tutti gli elementi della mesh e sommare i contributi locali per ottenere la matrice di stiffness globale $\mathbf{A}$ e il vettore $\mathbf{f}$. Si presti attenzione ad assemblare $\mathbf{A}$ come matrice sparsa.

Soluzione dell'esercizio 8

La soluzione si trova nel file `Assembly.jl`.

Esercizio 9

Assemblare la matrice di stiffness $\tilde{\mathbf{A}}$ e il load vector $\mathbf{f}$ per il problema di Poisson sul cerchio unitario $\Omega = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| \leq 1\}$, considerando il termine sorgente $f(x, y) = x$. Eseguire l'assemblaggio per tutti i gradi di libertà, sia interni che di bordo, utilizzando le funzioni `assemble_global` e `poisson_assemble_local!` implementate in precedenza.

Per verificare la correttezza della propria implementazione, effettuare i seguenti controlli:

1. verificare che $\tilde{\mathbf{A}}$ sia simmetrica;
2. controllare che $\tilde{\mathbf{A}}$ sia semidefinita positiva;
3. accertarsi che il nucleo di $\tilde{\mathbf{A}}$ sia unidimensionale e generato dal vettore costante, corrispondente alla funzione costante. Perché accade questo?
4. confrontare la matrice di stiffness e il vettore dei termini noti con quelli ottenuti utilizzando il pacchetto `Gridap.jl`. Il codice per l'assemblaggio con `Gridap` è disponibile in `ex04_1.jl`.

Soluzione dell'esercizio 9

La soluzione si trova nel file `ex04_1.jl`.