Elementi Finiti - Esercitazione 3
Generalizzazione della quadratura numerica

Prof. Giancarlo Sangalli   Ivan Bioli

31 Marzo 2025

Faremo uso della quadratura numerica anche nell'implementazione del metodo agli Elementi Finiti, quindi è utile generalizzare le formule di quadratura discusse sopra. L'idea è definire una formula di quadratura sull'elemento di riferimento $\hat{T}$ e poi applicare un push-forward a ciascun elemento $T \in \mathcal{T}_h$.

Consideriamo una formula di quadratura su $\hat{T}$ basata sui punti di quadratura $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_q \in \hat{T}$ e sui pesi $w_1, \ldots, w_q \in \mathbb{R}$. Questa approssima gli integrali su $\hat{T}$ come:

$$\int_{\hat{T}} u(\hat{\mathbf{x}}) \mathrm{d}\hat{\mathbf{x}} \approx \sum_{i=1}^{q} w_i \, u(\hat{\mathbf{p}}_i).$$

Ora vogliamo fare il push-forward di questa formula di quadratura a un triangolo generico $T \in \mathcal{T}_h$ con vertici $\mathbf{v}_T^1, \mathbf{v}_T^2, \mathbf{v}_T^3$. Poiché T è ottenuto come immagine di $\hat{T}$ tramite la mappa affine:

$$F(\hat{\mathbf{x}}) = \mathbf{B}\hat{\mathbf{x}} + \mathbf{a} \quad \text{dove} \quad \mathbf{B} = \begin{bmatrix} \mathbf{v}_T^2 - \mathbf{v}_T^1 & \mathbf{v}_T^3 - \mathbf{v}_T^1 \end{bmatrix}, \quad \mathbf{a} = \mathbf{v}_T^1, \tag{1}$$

possiamo approssimare l'integrale su T facendo il pull-back all'elemento di riferimento $\hat{T}$, cioè con il cambio di variabili $\mathbf{x} = F(\hat{\mathbf{x}})$. Sia u una funzione definita su T, allora:

$$\begin{aligned}\int_T u(\mathbf{x}) \mathrm{d}\mathbf{x} &= \int_T (u \circ F) \left(F^{-1}(\mathbf{x})\right) \mathrm{d}\mathbf{x} = \int_{\hat{T}} (u \circ F)(\hat{\mathbf{x}}) |\det JF(\hat{\mathbf{x}})| \mathrm{d}\hat{\mathbf{x}} \\ &= \int_{\hat{T}} (u \circ F)(\hat{\mathbf{x}}) |\det \mathbf{B}| \mathrm{d}\hat{\mathbf{x}} \approx \sum_{i=1}^{q} w_i \cdot |\det \mathbf{B}| \cdot (u \circ F)(\hat{\mathbf{p}}_i) \\ &= |\det \mathbf{B}| \cdot \sum_{i=1}^{q} w_i \, u(\mathbf{p}_i), \quad \text{dove} \quad \mathbf{p}_i = F(\hat{\mathbf{p}}_i) = \mathbf{B}\hat{\mathbf{p}}_i + \mathbf{a}.\end{aligned} \tag{2}$$

Dunque, il push-forward della formula di quadratura dall'elemento di riferimento a un triangolo generico comporta:

- la moltiplicazione dei pesi w_i per $|\det \mathbf{B}| = \text{area}(T)/\text{area}(\hat{T})$;
- il push-forward dei punti di quadratura $\hat{\mathbf{p}}_i$ tramite F, ottenendo $\mathbf{p}_i$.

Applicando lo stesso procedimento a tutti gli elementi della triangolazione, possiamo approssimare l'integrale $\int_\Omega u(\mathbf{x}) \mathrm{d}\mathbf{x}$ a partire da una regola di quadratura definita sull'elemento di riferimento.

Esercizio 1

Rivisitare gli esempi di quadrature dell'esercitazione precedente, inquadrandoli nella nuova metodologia. L'obiettivo è implementare la quadratura numerica sfruttando la trasformazione affine che mappa l'elemento di riferimento su ciascun triangolo della mesh.

Parte I

Determinare pesi e punti di quadratura delle regole Q_0, Q_1, Q_2 sull'elemento di riferimento $\hat{T}$.

Parte II

Definire una struttura dati in Julia per rappresentare una regola di quadratura su un triangolo, come segue:

Struct per la quadratura

```julia
struct TriQuad
    name::String
    order::Integer
    points::Matrix
    weights::Array
end
```

dove:

- `name` è il nome della regola di quadratura (ad esempio, `"Q0"`, `"Q1"`, `"Q2"`);
- `order` è il grado di precisione della quadratura;
- `points` è una matrice $2 \times q$ che contiene le coordinate dei punti di quadratura sull'elemento di riferimento;
- `weights` è un vettore di lunghezza q che contiene i pesi associati ai punti di quadratura.

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

Parte III

Definire una struttura dati per rappresentare una mesh triangolare:

Struct per la mesh

```julia
mutable struct Mesh
    const T::Matrix{Tp} where {Tp<:Integer}
    const p::Matrix{Tp} where {Tp<:Real}
    ak
    Bk
    detBk
end

function Mesh(T::Matrix{Tp} where {Tp<:Integer}, p::Matrix{Tp} where {Tp<:Real})
    return Mesh(T, p, nothing, nothing, nothing)
end
```

In questa struttura:

- `T` è la matrice di connettività;
- `p` è la matrice contenente le coordinate dei nodi della mesh;
- `Bk` e `ak` memorizzano, rispettivamente, la matrice e il vettore della trasformazione affine F che mappa l'elemento di riferimento $\hat{T}$ su ciascun triangolo T della mesh;
- `detBk` contiene il determinante della matrice $\mathbf{B}$ della trasformazione affine.

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

Parte IV

Implementare la funzione

```
function get_Bk!(mesh::Mesh)
```

che, data una mesh `mesh`, calcola, per ogni triangolo della mesh, la matrice $\mathbf{B}$ e il vettore $\mathbf{a}$ della trasformazione affine (1). La funzione deve restituire due array contenenti, rispettivamente, tutte le matrici $\mathbf{B}$ (taglia $2 \times 2 \times N_{\text{tri}}$) e tutti i vettori $\mathbf{a}$ (taglia $2 \times N_{\text{tri}}$) per l'intera mesh. Inoltre, questi array devono essere salvati nei campi `mesh.Bk` e `mesh.ak`. Si eviti di ricalcolare i valori più volte per la stessa mesh. Potete trovare un codice da completare in `Meshing.jl`.

Parte V

Implementare la funzione

```
function get_detBk!(mesh::Mesh)
```

che, data una mesh `mesh`, calcola, per ogni triangolo della mesh, il determinante della matrice $\mathbf{B}$ della trasformazione affine (1). La funzione deve restituire un array contenente tutti i determinanti per l'intera mesh. Inoltre, questo array deve essere salvato nel campo `mesh.detBk`. Si eviti di ricalcolare i valori più volte per la stessa mesh. Potete trovare un codice da completare in `Meshing.jl`.

Parte VI

Implementare la funzione

```
function Quadrature(u, mesh::Mesh, ref_quad::TriQuad)
```

che, data:

- una funzione u,
- una triangolazione `mesh`,
- una regola di quadratura `ref_quad` definita sull'elemento di riferimento,

approssima l'integrale di u sull'intero dominio. Potete trovare un codice da completare in `Quadrature_adv.jl`.

Parte VII

Testare che i risultati ottenuti con la funziona `Quadrature` coincidano con quelli della scorsa esercitazione.

Soluzione dell'esercizio 1

La regola di quadratura Q_0 ha come unico punto di quadratura il baricentro, con peso pari all'area del triangolo. Pertanto, per l'elemento di riferimento $\hat{T}$ di vertici $\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$, $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$, $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ abbiamo

$$\hat{\mathbf{p}}_1 = \begin{bmatrix}\frac{1}{3}\\\frac{1}{3}\end{bmatrix}, \qquad w_1 = \frac{1}{2}.$$

La regola di quadratura Q_1 ha come punti di quadratura i vertici, tutti con peso $\frac{1}{3}|T|$. Dunque:

$$\hat{\mathbf{p}}_1 = \begin{bmatrix}0\\0\end{bmatrix}, \hat{\mathbf{p}}_2 = \begin{bmatrix}1\\0\end{bmatrix}, \hat{\mathbf{p}}_2 = \begin{bmatrix}0\\1\end{bmatrix}, \qquad w_1 = w_2 = w_3 = \frac{1}{6}.$$

La regola di quadratura Q_2 ha come punti di quadratura i punti medi dei lati, tutti con peso $\frac{1}{3}|T|$. Dunque:

$$\hat{\mathbf{p}}_1 = \begin{bmatrix}\frac{1}{2}\\0\end{bmatrix}, \hat{\mathbf{p}}_2 = \begin{bmatrix}\frac{1}{2}\\\frac{1}{2}\end{bmatrix}, \hat{\mathbf{p}}_2 = \begin{bmatrix}0\\\frac{1}{2}\end{bmatrix}, \qquad w_1 = w_2 = w_3 = \frac{1}{6}.$$

La struct `Mesh` e le funzioni `get_Bk!`, `get_detBk!`, sono implementate in `Meshing.jl`. La struct `TriQuad` e la funzione `Quadrature` sono implementati in `Quadrature_adv.jl`. Alcuni test numerici sono presenti in `ex03_1.jl` e `ex03_2.jl`.