Alma Mater Studiorum · Università di Bologna

**Facoltà di Scienze Matematiche, Fisiche e Naturali**

Corso di Laurea in Matematica

# CALCOLO E VISUALIZZAZIONE DELLE FUNZIONI DI TAGLIA DI CURVE SPLINE

Tesi di laurea in Topologia Algebrica

**Presentata da:**
Barbara Di Fabio

**Relatore:**
Prof. Massimo Ferri

**Correlatori:**
Prof. Ilio Galligani
Dr. Patrizio Frosini

**Parole chiave:** Pattern Recognition, Curve di Bézier, Punti Critici, Funzioni Polinomiali, Modellazione 2D

Sessione II

Anno Accademico 2003–2004

*Alla mia famiglia,*
*a Filippo e a Runny*

# Indice

# Introduzione

La Teoria della Taglia è un nuovo strumento matematico, ancora in via di sperimentazione, nato come nuovo approccio al problema del riconoscimento e del confronto delle forme.
La sua struttura è basata su due concetti strettamente correlati: le distanze naturali di taglia e le funzioni di taglia.
Le prime rappresentano un metodo per misurare il grado di somiglianza tra due varietà topologiche omeomorfe, su ognuna delle quali è definita una funzione a valori in $\mathbb{R}^k$, detta funzione misurante. L'idea di base per calcolarla è molto semplice: si misura l'entità della variazione della funzione misurante quando ci si sposta dalla prima varietà alla seconda tramite un omeomorfismo.
Le funzioni di taglia, più facili da calcolare, forniscono proprietà metriche e topologiche delle varietà e limiti inferiori alle distanze naturali di taglia. Il loro supporto è determinato completamente dai punti critici della varietà rispetto alla funzione misurante definita e quindi, sono questi a fornire tutte le informazioni necessarie sull'oggetto in esame.
La possibilità di applicare i risultati raggiunti fino ad ora da questa teoria anche allo studio di forme che presentano diversi gradi di regolarità, ne permette il consolidamento e l'aumento di generalità.
Oggigiorno, uno degli strumenti più potenti per fare approssimazione di forma nell'ambito della computer grafica è rappresentato dalle curve spline, cioè curve polinomiali a tratti definite da un numero finito di punti nel piano, detti punti di controllo. La caratteristica strutturale di queste curve, così, le

rende enormemente flessibili e la possibilità di controllarle localmente, ci permette di poter operare in modo efficiente con polinomi al più di terzo grado; quest'ultima proprietà le rende semplici da calcolare e numericamente stabili.

In questo lavoro di tesi ci siamo avvicinati al problema della teoria della taglia su curve spline, scegliendone una classe molto regolare. Abbiamo preso in esame curve aperte di secondo grado con partizione nodale uniforme associata e su queste abbiamo definito la funzione misurante altezza, che ad ogni punto della spline associa la sua ordinata nel piano.
Nel primo capitolo sono presentate le nozioni, le definizioni e le proprietà principali relative alla teoria della taglia e alla teoria delle spline.
Nel secondo capitolo sono stati formalizzati i risultati pervenuti dalla ricerca dei punti critici di spline di secondo grado rispetto alla funzione altezza.
Nel terzo capitolo vengono presentati la struttura e i metodi di funzionamento del programma di visualizzazione, *Size & Spline*, che mostra le variazioni in tempo reale della funzione di taglia di una curva spline sottoposta a modifiche di forma interattive.
In appendice viene presentato il codice del programma sviluppato.

# Capitolo 1

# Preliminari Matematici

## 1.1 La Teoria della Taglia

Il confronto di forme di oggetti è uno dei principali obiettivi della visione artificiale.
La Teoria della Taglia è un nuovo strumento matematico che cerca di rispondere a tale esigenza ed è tuttora in via di sperimentazione.
Le idee fondamentali che la caratterizzano sono i concetti di distanza naturale di taglia e di funzione di taglia. Entrambi rappresentano un modo per misurare quanto le forme di due spazi topologici compatti si assomiglino. Risultano intrinsecamente correlati e costituiscono concetti modulari, nel senso che dipendono dalla scelta arbitraria di particolari funzioni (dette funzioni misuranti), che possono essere definite in modo tale da ottenere invarianza sotto quei tipi di trasformazioni di cui si richiede di preservare la forma in ogni specifico contesto.

### 1.1.1 Distanze naturali di taglia: alcune definizioni e risultati

In questo paragrafo consideriamo l'insieme *Size* di tutte le coppie $(\mathcal{M}, \varphi)$ (dette *coppie di taglia*), dove $\mathcal{M}$ è uno spazio topologico compatto e $\varphi$ è una

funzione continua da $\mathcal{M}$ ad $\mathbb{R}$, insieme dei numeri reali, chiamata *funzione misurante*.
Il nostro proposito è quello di definire una distanza che ci permetta di misurare quanto siano simili le forme di due spazi topologici $\mathcal{M}$ ed $\mathcal{N}$ messi a confronto; lo faremo rispetto alle funzioni continue $\varphi$ e $\psi$, scelte arbitrariamente.

**Definizione 1.1** Siano $(\mathcal{M}, \varphi)$, $(\mathcal{N}, \psi)$ due coppie di taglia e sia $H(\mathcal{M}, \mathcal{N})$ l'insieme degli omeomorfismi da $\mathcal{M}$ ad $\mathcal{N}$. Sia $\Theta$ la funzione che ad ogni omeomorfismo $f \in H(\mathcal{M}, \mathcal{N})$ associa il numero reale

$$\Theta(f) = \max_{P \in \mathcal{M}} |\varphi(P) - \psi(f(P))|$$

Chiameremo $\Theta$ la *misura naturale di taglia* in $H(\mathcal{M}, \mathcal{N})$, rispetto alle funzioni misuranti $\varphi$ e $\psi$.

In altre parole, $\Theta$ misura quanto $f$ cambia i valori assunti dalla funzione misurante.

**Proposizione 1.1** La funzione $\Sigma : \mathit{Size} \times \mathit{Size} \longrightarrow \mathbb{R} \cup \{\infty\}$, definita ponendo $\Sigma((\mathcal{M}, \varphi), (\mathcal{N}, \psi)) = \inf_{f \in H(\mathcal{M}, \mathcal{N})} \Theta(f)$ se $H(\mathcal{M}, \mathcal{N}) \neq \emptyset$ e $+\infty$ altrimenti, è una pseudometrica su *Size*.

**Definizione 1.2** La metrica $\sigma$ indotta dalla pseudometrica $\Sigma$ sarà chiamata la *distanza naturale di taglia* in $\mathit{Size}/\backsimeq$, dove $\backsimeq$ denota la relazione di equivalenza definita ponendo $(\mathcal{M}, \varphi) \backsimeq (\mathcal{N}, \psi)$ se e solo se $\Sigma((\mathcal{M}, \varphi), (\mathcal{N}, \psi)) = 0$. La classe di equivalenza di $(\mathcal{M}, \varphi)$ sarà denotata con il simbolo $[(\mathcal{M}, \varphi)]$.

**Definizione 1.3** Chiameremo *ottimale in* $H(\mathcal{M}, \mathcal{N})$ ogni omeomorfismo $f \in H(\mathcal{M}, \mathcal{N})$ tale che $\sigma([(\mathcal{M}, \varphi)], [(\mathcal{N}, \psi)]) = \Theta(f)$.

Vogliamo sottolineare che un omeomorfismo ottimale non esiste in generale, anche nei casi in cui $\mathcal{M}$ ed $\mathcal{N}$ siano varietà regolari, compatte e senza bordo e $\varphi$ e $\psi$ siano funzioni misuranti regolari. Comunque, se assumiamo

una tale ipotesi (in particolare che $\mathcal{M}$ ed $\mathcal{N}$ siano varietà di classe $C^2$ e $\varphi$ e $\psi$ siano funzioni misuranti di classe $C^1$), otteniamo il seguente risultato:

**Teorema 1.1** Supponiamo che esista un omeomorfismo ottimale in $H(\mathcal{M}, \mathcal{N})$. Allora la distanza naturale di taglia tra $[(\mathcal{M}, \varphi)]$ ed $[(\mathcal{N}, \psi)]$ è la distanza euclidea tra un valore critico di $\varphi$ ed un valore critico di $\psi$.

### 1.1.2 Funzioni di taglia: definizioni e proprietà

In generale, le distanze naturali di taglia sono difficili da calcolare, poiché richiedono lo studio di tutti i tipi di omeomorfismi tra due spazi topologici compatti. D'altra parte, esse permettono il confronto tra spazi topologici compatti rispetto a funzioni misuranti date, in un modo molto potente, e ne quantificano la differenza. Così abbiamo bisogno di uno strumento per ottenere facilmente informazioni sulle distanze naturali di taglia senza doverle calcolare direttamente. Introduciamo perciò il concetto di funzione di taglia. In più, le funzioni di taglia sono utili per il confronto delle forme anche indipendentemente dalle distanze naturali di taglia.
Nelle seguenti definizioni assumeremo che sia stata fissata una coppia di taglia $(\mathcal{M}, \varphi)$.

**Definizione 1.4** Per ogni $y \in \mathbb{R}$ definiamo una relazione $\approxeq_{\varphi \leq y}$ in $\mathcal{M}$ ponendo $P \approxeq_{\varphi \leq y} Q$ $(P, Q \in \mathcal{M})$ se e solo se o $P = Q$ o esiste un cammino continuo $\gamma : [0,1] \longrightarrow \mathcal{M}$ tale che $\gamma(0) = P$, $\gamma(1) = Q$ e $\varphi(\gamma(\tau)) \leq y$ per ogni $\tau \in [0,1]$. In questo secondo caso, diremo che $P$, $Q$ sono $(\varphi \leq y)$-*omotopi* e chiameremo $\gamma$ una $(\varphi \leq y)$-*omotopia da* $P$ *a* $Q$.

**Osservazione 1.1** Si dimostra facilmente che $\approxeq_{\varphi \leq y}$ è una relazione di equivalenza su $\mathcal{M}$ per ogni $y \in \mathbb{R}$.

**Definizione 1.5** Per ogni $x \in \mathbb{R}$, denoteremo con $\mathcal{M}\langle \varphi \leq x \rangle$ l'insieme $\{P \in \mathcal{M} : \varphi(P) \leq x\}$.

**Definizione 1.6** Si consideri la funzione $l_{(\mathcal{M},\varphi)} : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{N} \cup \{\infty\}$ definita ponendo $l_{(\mathcal{M},\varphi)}(x, y)$ uguale al numero (finito o infinito) di classi di equivalenza in cui $\mathcal{M}\langle\varphi \leq x\rangle$ è diviso dalla relazione di equivalenza $\approxeq_{\varphi\leq y}$ . Una tale funzione sarà chiamata la *funzione di taglia associata alla coppia di taglia* $(\mathcal{M}, \varphi)$.

**Osservazione 1.2** Quando $x \leq y$, le funzioni di taglia hanno una semplice interpretazione geometrica: $l_{(\mathcal{M},\varphi)}(x, y)$ è uguale al numero di componenti connesse per archi di $\mathcal{M}\langle\varphi \leq y\rangle$ contenenti almeno un punto di $\mathcal{M}\langle\varphi \leq x\rangle$.

**Principali proprietà delle funzioni di taglia** $l_{(\mathcal{M},\varphi)}$

(1) $l_{(\mathcal{M},\varphi)}(x, y)$ è non decrescente in $x$ e non crescente in $y$.

(2) $l_{(\mathcal{M},\varphi)}(x, y)$ è finita per $x < y$ (sotto l'ipotesi che $\mathcal{M}$ è un compatto non vuoto e localmente connesso per archi).

(3) $l_{(\mathcal{M},\varphi)}(x, y) = 0$ per ogni $x < \min_{P\in\mathcal{M}} \varphi(P)$.

(4) $l_{(\mathcal{M},\varphi)}(x, y)$ è uguale al numero di componenti connesse per archi di $\mathcal{M}$ per ogni $x, y > \max_{P\in\mathcal{M}} \varphi(P)$.

(5) $l_{(\mathcal{M},\varphi)}(x, y) = \infty$ per ogni $x, y$ tali che esiste un punto non isolato $Q \in \mathcal{M}$ per cui $x > \varphi(Q)$ e $y < \varphi(Q)$.

Completiamo questo sommario delle proprietà dell funzioni di taglia enunciando un risultato che ci aiuta a localizzare i punti di discontinuità delle funzioni di taglia $l_{(\mathcal{M},\varphi)}$ nel caso in cui $\mathcal{M}$ sia una sottovarietà chiusa dello spazio euclideo (cioè, compatta e senza bordo).

**Teorema 1.2** Supponiamo che $\mathcal{M}$ sia $C^2$ e che la funzione misurante $\varphi$ sia $C^1$. Allora, se $(x, y)$ è un punto di discontinuità per la funzione di taglia $l_{(\mathcal{M},\varphi)}$ ed $x < y$, o $x$ oppure $y$ o entrambi sono valori critici per $\varphi$.

### 1.1.3 Il legame tra funzioni di taglia e distanze naturali di taglia

Il punto chiave della teoria della taglia è che le distanze naturali di taglia e le funzioni di taglia risultano strettamente correlate. Questo ci permette di ottenere informazioni sulle prime (più potenti ma intrinsecamente più difficili da calcolare) studiando le seconde (facilmente calcolabili).
Questa affermazione è una conseguenza dei seguenti due teoremi.

**Teorema 1.3** Se $\sigma([(\mathcal{M},\varphi)],[(\mathcal{N},\psi)]) < \varepsilon$, allora per ogni $x, y \in \mathbb{R}$ ed ogni $h \geq \varepsilon$, valgono le seguenti disuguaglianze:

$$l_{(\mathcal{M},\varphi)}(x-h, y+h) \leq l_{(\mathcal{N},\psi)}(x,y) \leq l_{(\mathcal{M},\varphi)}(x+h, y-h)$$

**Teorema 1.4** Si supponga che esistano $\overline{x}, \overline{y}, \widetilde{x}, \widetilde{y} \in \mathbb{R}$ per cui $l_{(\mathcal{M},\varphi)}(\overline{x},\overline{y}) > l_{(\mathcal{M},\varphi)}(\widetilde{x},\widetilde{y})$. Allora, si ha che:

$$\sigma([(\mathcal{M},\varphi)],[(\mathcal{N},\psi)]) \geq \min\{\widetilde{x}-\overline{x}, \overline{y}-\widetilde{y}\}$$

Vogliamo sottolineare che l'ultimo teorema ci permette di ottenere un limite dal basso per le distanze naturali di taglia tramite la conoscenza delle funzioni di taglia in due punti.

### 1.1.4 Calcolo delle funzioni di taglia

In questo paragrafo, vogliamo dare una tecnica per calcolare la funzione $l_{(\mathcal{M},\varphi)}(x,y)$. D'ora in avanti assumeremo che $\mathcal{M}$ sia un sottoinsieme di $\mathbb{R}^m$ compatto e localmente connesso per archi e che la funzione misurante $\varphi$ sia la restrizione ad $\mathcal{M}$ di una funzione continua $\overline{\varphi} : \mathbb{R}^m \longrightarrow \mathbb{R}$. Inoltre, denoteremo con $\omega(\delta)$ *il modulo di continuità* della funzione misurante $\overline{\varphi}$ (cioè $\omega(\delta) = \sup\{|\overline{\varphi}(P) - \overline{\varphi}(Q)| : P, Q \in \mathbb{R}^m, \|P-Q\| < \delta\}$. Vogliamo approssimare l'insieme considerato $\mathcal{M}$ e la corrispondente funzione $l_{(\mathcal{M},\varphi)}(x,y)$ con un insieme finito $\mathcal{P}$ ed una funzione $l_{approx}$ rispettivamente. La funzione $l_{approx}$ sarà correlata a $l_{(\mathcal{M},\varphi)}(x,y)$ e risulterà più semplice da calcolare.

**Definizione 1.7** Sia $\mathcal{P} = \{P_0, P_1, ...P_h\}$ un insieme finito di punti di $\mathbb{R}^m$, e denotiamo con $\mathcal{B}_\delta$ l'insieme delle $h+1$ palle aperte $B(P_i, \delta)$ di raggio $\delta$ con centri i punti di $\mathcal{P}$. Assumiamo che $\mathcal{B}_\delta$ verifichi le seguenti proprietà:

(1) $\mathcal{M}$ è contenuta in $\bigcup_{i=0}^{h} B(P_i, \delta)$

(2) Per ogni indice $0 \leq i \leq h$, $B(P_i, \delta) \cap \mathcal{M}$ è un insieme connesso per archi non vuoto.

Chiamiamo $\mathcal{B}_\delta$ un $\delta$-*ricoprimento* di $\mathcal{M}$. L'insieme $\mathcal{P}$ sarà chiamato l'*insieme dei centri* di $\mathcal{B}_\delta$. Denotiamo con $\sim$ la seguente relazione sull'insieme $\mathcal{P}$: $P_i \sim P_j$ se $(B(P_i, \delta) \cup B(P_j, \delta)) \cap \mathcal{M}$ è una componente connessa per archi.

**Osservazione 1.3** Ovviamente $\sim$ è una relazione riflessiva e transitiva.

D'ora in avanti assumeremo che sia dato un $\delta$-ricoprimento di $\mathcal{M}$, $\mathcal{B}_\delta$, e denotiamo con $\mathcal{P}$ l'insieme $\{P_0, P_1, ...P_h\}$ dei suoi centri.

**Definizione 1.8** Per ogni $x, y \in \mathbb{R}$ denotiamo con $\mathcal{P}\langle\overline{\varphi} \leq x\rangle$ l'inseme degli elementi di $\mathcal{P}$ su cui la funzione $\overline{\varphi}$ assume un valore non più grande di $x$, e con $\sim_{\overline{\varphi}\leq y}$ la relazione d'equivalenza su $\mathcal{P}$ definita nel seguente modo: se $P_a, P_b \in \mathcal{P}$ scriviamo $P_a \sim_{\overline{\varphi}\leq y} P_b$ nei casi in cui o $P_a = P_b$ oppure esiste una successione finita di punti $(P_{s(i)})_{i=0,...,r}$ in $\mathcal{P}\langle\overline{\varphi} \leq y\rangle$, tale che $P_{s(0)} = P_a$,$P_{s(r)} = P_b$, e per ogni indice $i$ con $0 \leq i \leq r-1$ abbiamo $P_{s(i)} \sim P_{s(i+1)}$. Questa relazione d'equivalenza viene chiamata $(\overline{\varphi} \leq y)$-*equivalenza relativa a* $\mathcal{B}_\delta$. Denotiamo con $l_{approx}(x, y)$ il numero di classi di equivalenza in cui $\mathcal{P}\langle\overline{\varphi} \leq x\rangle$ è diviso dalla $(\overline{\varphi} \leq y)$-equivalenza.

Ora possiamo dare due risultati che ci permettono di calcolare facilmente le funzioni di taglia.

**Teorema 1.5** Per ogni $x, y \in \mathbb{R}$ e per ogni $\overline{\omega} \geq \omega(\delta)$, con $x + \overline{\omega} \leq y - \overline{\omega}$, si ha:

$$l_{approx}(x - \overline{\omega}, y + \overline{\omega}) \leq l_{(\mathcal{M},\varphi)}(x, y) \leq l_{approx}(x + \overline{\omega}, y - \overline{\omega})$$

$$l_{(\mathcal{M},\varphi)}(x-\overline{\omega},y+\overline{\omega}) \leq l_{approx}(x,y) \leq l_{(\mathcal{M},\varphi)}(x+\overline{\omega},y-\overline{\omega})$$

**Corollario 1.1** Siano $\overline{x},\overline{y},b,c \in \mathbb{R}$ con $b,c \geq 0$ e $\overline{\omega} \geq \omega(\delta)$, con $\overline{x}+\overline{\omega} \leq \overline{y}-\overline{\omega}$. Se la funzione $l_{approx}$ prende lo stesso valore $v$ nei punti $(\overline{x}+\overline{\omega},\overline{y}-\overline{\omega})$ e $(\overline{x}-b-\overline{\omega},\overline{y}+c+\overline{\omega})$, allora si ha $l_{(\mathcal{M},\varphi)}(x,y)=v$, $\forall\,(x,y)$ nel rettangolo

$$R=\left\{(x,y)\in\mathbb{R}^2 : \overline{x}+b\leq x\leq\overline{x},\ \overline{y}\leq y\leq\overline{y}+c\right\}$$

.

### 1.1.5 Rappresentazione algebrica delle funzioni di taglia

Le discontinuità delle funzioni di taglia giocano un ruolo cruciale nei processi di "riconoscimento". Infatti, risulta che i punti di discontinuità trasportano tutte le informazioni più importanti contenute nelle funzioni di taglia riguardanti l'oggetto di studio. Il problema di riuscire a catturare le informazioni dateci dalle discontinuità è risolvibile nel caso in cui $\mathcal{M}$ sia unione finita di sottoinsiemi dello spazio euclideo compatti e localmente connessi per archi. Infatti, in tal caso, i punti di discontinuità di una funzione di taglia appartengono alla regione $\mathcal{S}_0=\{(x,y)\in\mathbb{R} : x<y\}$, che divide la parte del dominio di una funzione di taglia al di sopra della diagonale $\{(x,y)\in\mathbb{R}^2 : x=y\}$ in regioni triangolari sovrapposte e aventi tutte un lato sulla diagonale. Alcuni triangoli possono avere area infinita.
In generale, identificheremo ogni triangolo di area finita con un punto ed ogni triangolo con area infinita con una retta verticale.
Più precisamente, chiameremo *punto d'angolo* ogni punto $p=(x,y)$, con $x<y$, che soddisfi alla seguente proprietà:
se denotiamo con $\mu_{\alpha,\beta}$ il numero

$$l_{(\mathcal{M},\varphi)}(x+\alpha,y-\beta)-l_{(\mathcal{M},\varphi)}(x+\alpha,y+\beta)-l_{(\mathcal{M},\varphi)}(x-\alpha,y-\beta)+l_{(\mathcal{M},\varphi)}(x-\alpha,y+\beta)$$

allora deve valere:

$$\mu(p)\stackrel{def}{=}\min\{\mu_{\alpha,\beta} : \alpha>0,\beta>0,x+\alpha<y-\beta\}>0.$$

Chiameremo $\mu(p)$ la *molteplicità di p*.
Analogamente, chiameremo *retta d'angolo* ogni retta verticale $r : x = k\,(k \in \mathbb{R})$ per cui vale:

$$\mu(r) \stackrel{def}{=} \min_{\alpha>0,\, k+\alpha<y} l_{(\mathcal{M},\varphi)}(k+\alpha, y) - l_{(\mathcal{M},\varphi)}(k-\alpha, y) > 0.$$

Chiameremo $\mu(r)$ la *molteplicità di r*.
L'importanza dei punti d'angolo e delle rette d'angolo delle funzioni di taglia risiede nel fatto che il valore di una funzione di taglia in un qualsiasi punto che giace al di sopra della diagonale, è uguale alla somma delle molteplicità dei punti d'angolo e delle rette d'angolo che identificano i triangoli contenenti quel punto.
Cf. [eCL99] per maggiori dettagli.

## 1.2 Curve di Bézier

**Definizione 1.9** Una *curva di Bézier* è una curva polinomiale parametrica definita nel seguente modo:

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i \qquad a \le u \le b$$

dove le funzioni base, $\{B_{i,n}(u)\}$, sono le *funzioni polinomiali di Bernstein*, date da:

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} \frac{(u-a)^i (b-u)^{n-i}}{(b-a)^n}$$

e i coefficienti, $\{P_i\}$, sono chiamati *punti di controllo*.

D'ora in avanti considereremo $u \in [0, 1]$.
La scelta delle funzioni base determina le caratteristiche geometriche delle curve e superfici costruite come loro combinazione lineare. Le principali proprietà riguardanti i polinomi di Bernstein sono:

1. **Non negatività:** $B_{i,n}(u) \ge 0$ per ogni $i, n$ e $0 \le u \le 1$;

2. **Partizione dell'unità:** $\sum_{i=0}^{n} B_{i,n}(u) = 1$ per ogni $0 \leq u \leq 1$;

3. $B_{0,n}(0) = B_{n,n}(1) = 1$;

4. $B_{i,n}(u)$ ammette esattamente un punto di massimo sull'intervallo $[0, 1]$ e, precisamente, in $u = \frac{i}{n}$;

5. **Simmetria:** Per ogni $n$, l'insieme dei polinomi $\{B_{i,n}(u)\}$ è simmetrico rispetto ad $u = \frac{1}{2}$;

6. **Definizione ricorsiva:** $B_{i,n}(u) = (1-u)B_{i,n-1}(u) + uB_{i-1,n-1}(u)$; definiamo $B_{i,n}(u) \equiv 0$ se $i < 0$ oppure $i > n$;

7. **Derivate:**

$$B'_{i,n}(u) = \frac{dB_{i,n}(u)}{du} = n(B_{i-1,n-1}(u) - B_{i,n-1}(u))$$

con

$$B_{-1,n-1}(u) \equiv B_{n,n-1}(u) \equiv 0.$$

$$\begin{aligned} C(u) &= \frac{d\left(\sum_{i=0}^{n} B_{i,n}(u)P_i\right)}{du} = \sum_{i=0}^{n} B'_{i,n}(u)P_i \\ &= \sum_{i=0}^{n} n(B_{i-1,n-1}(u) - B_{i,n-1}(u))P_i \\ &= n\sum_{i=0}^{n-1} B_{i,n-1}(u))(P_{i+1} - P_i) \end{aligned}$$

da cui possiamo ottenere facilmente le formule delle derivate nei punti estremi:

$$C'(0) = n(P_1 - P_0)$$
$$C'(1) = n(P_n - P_{n-1})$$

## 1.3 Funzioni B-Spline

Le curve di Bézier sono un tipico esempio di curve difinite da un'unica funzione polinomiale, in quanto ogni funzione base è non nulla su tutto il dominio

parametrico, e quindi apporta un contributo non nullo all'andamento di tutta la curva. Questo fatto, proprio delle curve definite da una sola funzione polinomiale, le rende spesso inadatte come approssimanti di forma di altre curve. Non è possibile, infatti, modificarle localmente; non possono essere utilizzate su curve che presentano diversi gradi di differenziabilità; infine, si richiede almeno grado $n-1$ perchè la funzione polinomiale interpoli $n$ punti dati (non allineati). Quest'ultimo fatto rappresenta un problema quando $n$ è molto grande; infatti si crea un fenomeno di rumore, noto come "fenomeno di Runge", dovuto alla presenza delle oscillazioni che caratterizzano i polinomi con gradi elevati.
La soluzione a questi problemi si avvale dell'uso dei polinomi a tratti; la curva in esame viene approssimata o interpolata localmente con polinomi di grado basso. L'inconveniente principale di questi, però, sta nel poter perdere continuità nei punti di raccordo tra due polinomi contigui.

### 1.3.1 Definizione e proprietà delle funzioni spline

Denotiamo con

$$P_m = \{p : [a,b] \longrightarrow \mathbb{R},\, p(x) = \sum_{i=0}^{m} a_i x^i,\, a_0, ..., a_m \in \mathbb{R}\}$$

lo spazio vettoriale dei polinomi di grado $\leq m$.

**Definizione 1.10** Definiamo lo *spazio dei polinomi a tratti di grado m*:

$$PP_m = \{f|\ \exists\, p_0, ..., p_k \in P_m : f(x) = p_i(x), \forall\, x \in I_i,\, i = 0, ..., k\},$$

dove abbiamo considerato una partizione di $[a,b]$, $\Delta = \{x_i\}_{i=1,...,k}$

$$a = x_0 < x_1 < ... < x_k < x_{k+1} = b$$

e i $k+1$ sottointervalli:

$$I_i = [x_i, x_{i+1}), \quad i = 0, ..., k$$
$$I_k = [x_k, x_{k+1}].$$

**Definizione 1.11** Definiamo lo *spazio delle spline polinomiali a nodi semplici* di grado $m$ con $k$ nodi fissati

$$a \equiv x_0 < x_1 < ... < x_k < x_{k+1} \equiv b$$

come

$$S_m(\Delta) = PP_m \cap C^{m-1}_{[a,b]}$$

ossia:

$$S_m(x_1, ..., x_k) = \{s \in C^{m-1}_{[a,b]} : s\,|_{I_i} \in P_m,\, i = 0, ..., k\}.$$

**Osservazione 1.4** Con l'introduzione delle funzioni spline, si corregge la non regolarità dei polinomi a tratti, ottenendo funzioni derivabili con continuità fino all'ordine $m-1$, che è la massima regolarità che si può richiedere ai polinomi a tratti, per non ricadere nel caso di un unico polinomio definito su tutto $[a, b]$.

**Osservazione 1.5** Risulta che:

$$P_m \subset S_m(\Delta) \subset PP_m(\Delta).$$

Esistono però altri spazi di funzioni che presentano caratteristiche di regolarità, ma non così elevate come per lo spazio $S_m(\Delta)$. Questi sono gli spazi delle spline a nodi multipli.

**Definizione 1.12** Sia $[a, b]$ un intervallo chiuso e limitato, $\Delta = \{x_i\}_{i=1,...,k}$ un insieme di punti tale che $a = x_0 < x_1 < ... < x_k < x_{k+1} = b$; consideriamo la partizione di $[a, b]$ indotta dall'insieme $\Delta$ nei sottointervalli $I_i$, $i = 0, ..., k$, come sopra. Sia poi $m$ un intero positivo, $M = (m_1, m_2, ..., m_k)$ un vettore di interi positivi tali che $1 \leq m_i \leq m + 1, \forall\, i = 1, ..., k$.
Si definisce l'*insieme delle spline di grado m con nodi* $x_1, ..., x_k$ *di molteplicità* $m_1, ..., m_k$, come:

$$S(P_m, M, \Delta) = \{s(x) \mid \exists\, s_0(x), ..., s_k(x) \in P_m :$$

(1) $s(x) = s_i(x)$, per $x \in I_i$, $i = 0, ..., k$

(2) condizione di continuità sui nodi:
$D^l_{s_{i-1}}(x_i) = D^l_{s_i}(x_i)$, per $l = 0, ..., m - m_i$, $i = 1, ..., k\}$

**Teorema 1.6** L'insieme delle spline a nodi multipli $S(P_m, M, \Delta)$ è uno spazio di funzioni di dimensione $m + r + 1$, dove:

$$r = \sum_{i=1}^{k} m_i$$

Ogni elemento $s(x)$ dello spazio $S(P_m, M, \Delta)$, cioè ogni funzione spline, può essere rappresentato nel seguente modo:

$$s(x) = \sum_{i=0}^{n} c_i \varphi_i,$$

dove $n = m+r$ e $\{\varphi_0(x), \varphi_1(x), ..., \varphi_n(x)\}$ è una base dello spazio $S(P_m, M, \Delta)$; $\varphi_0(x), \varphi_1(x), ..., \varphi_n(x)$ sono dette funzioni base di $S(P_m, M, \Delta)$.
Una base stabile dal punto di vista computazionale è costituita dalle funzioni B-spline normalizzate.

**Definizione 1.13** L'insieme $\Delta^* = \{u_i\}_{i=0,...,2m+r+1}$, $r = \sum_{i=1}^{k} m_i$, si chiama *partizione nodale estesa associata ad* $S(P_m, M, \Delta)$ se e solo se:

(1) $u_0 \leq u_1 \leq ... \leq u_{2m+r+1}$;

(2) $u_m \equiv a$; $u_{m+r+1} \equiv b$;

(3) $(u_{m+1} \leq ... \leq u_{m+r}) \equiv (\underbrace{x_1 = ... = x_1}_{m_1 volte} < ... < \underbrace{x_k = ... = x_k}_{m_k volte})$.

**Definizione 1.14** Sia $\Delta^*$ la partizione estesa associata allo spazio $S(P_m, M, \Delta)$. Definiamo l'*insieme delle funzioni B-spline normalizzate*

$$\{N_{i,m}(u)\}_{i=0,...,n},\ u \in [a, b]$$

mediante la seguente formula ricorsiva:

$$N_{i,0}(u) = \begin{cases} 1, & u_i \leq u < u_{i+1} \\ 0, & altrimenti \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \quad p = 1, ..., m.$$

Si noti che:

- $N_{i,0}(u)$ è una funzione a scala, uguale a 0 dappertutto, eccetto sull'intervallo semiaperto $[u_i, u_{i+1})$;

- Per $p > 0$, $N_{i,p}(u)$ è una combinazione lineare di due funzioni base di grado $p - 1$;

- Il calcolo di un insieme di funzioni base richiede la specificazione di un vettore di nodi $\Delta^*$ e il grado $m$;

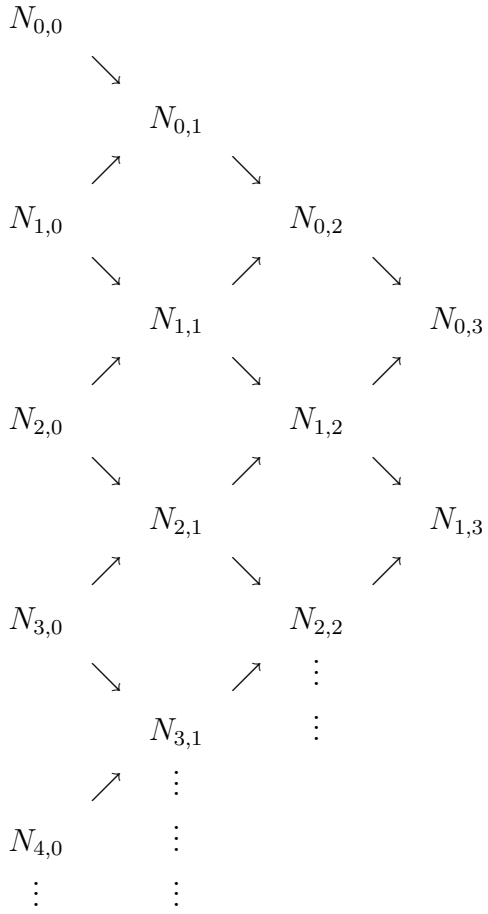
- La formula per determinarle può dare il quoziente 0/0; in tal caso lo definiamo 0;

- Le $N_{i,p}(u)$ sono polinomiali a tratti definite sull'intera retta reale;

- L'intervallo semiaperto $[u_i, u_{i+1})$ è detto l'*i-esimo intervallo nodale*; esso può avere lunghezza nulla, dal momento che i nodi possono essere coincidenti;

- Il calcolo delle funzioni di grado $m$ genera una tavola triangolare tron-

cata:




### 1.3.2 Proprietà delle funzioni base B-spline

Ora elenchiamo le principali proprietà di cui godono le funzioni base B-spline; sono queste, infatti, a determinare le molte desiderate caratteristiche delle curve e superfici B-spline.
Assumiamo che il loro grado sia $m$ e il vettore dei nodi $\Delta^* = \{u_0, ..., u_{2m+r+1}\}$.

1. **Proprietà di supporto locale:** $N_{i,m}(u) = 0$ se $u$ è al di fuori dell'intervallo $[u_i, u_{i+m+1})$;

2. In ogni intervallo nodale dato, $[u_j, u_{j+1})$, al più $m + 1$ delle $N_{i,m}$ sono non nulle, cioè le funzioni $N_{j-m,m}, ..., N_{j,m}$;

3. **Proprietà di non negatività:** $N_{i,m}(u) \geq 0$, $\forall\, i, m, u$;

4. **Proprietà di partizione dell'unità:** Per un intervallo nodale arbitrario, $[u_i, u_{i+1})$, $\sum_{j=i-m}^{i} N_{j,m}(u) = 1$, $\forall\, u \in [u_i, u_{i+1})$;

5. Tutte le derivate di $N_{i,m}(u)$ esistono all'interno di un intervallo nodale. In corrispondenza di un nodo $u_i$, $N_{i,m}(u)$ risulta $m - m_i$ volte differenziabile in modo continuo, se $m_i$ è la molteplicità del nodo;

6. A parte il caso $m = 0$, $N_{i,m}(u)$ ammette esattamente un valore massimo.

### 1.3.3 Definizione e proprietà delle curve B-spline

**Definizione 1.15** Una *curva B-spline* è definita come:

$$C(u) = \sum_{i=0}^{n} N_{i,m}(u) P_i, \;\; u \in [a, b]$$

dove i $P_i$ sono chiamati *punti di controllo* e le $\{N_{i,m}(u)\}$ sono le funzioni base B-spline di grado $m$ definite su un vettore nodale non periodico a nodi aggiuntivi coincidenti:

$$\Delta^* = \{\underbrace{a, ..., a}_{m+1}, u_{m+1}, ..., u_{m+r}, \underbrace{b, ..., b}_{m+1}\}$$

Salvo indicazione contraria, d'ora in poi assumeremo $a = 0$, $b = 1$.
Il poligono formato dai $P_i$ è detto *poligono di controllo*.

Ora elenchiamo alcune proprietà delle curve B-spline.

1. Se $n = m$ e $\Delta^* = \{0, ..., 0, 1, ..., 1\}$, allora $C(u)$ è una curva di Bézier;

2. $C(u)$ è una curva polinomiale a tratti; il grado $m$, il numero di punti di controllo, $n + 1$, e il numero di nodi, $2(m + 1) + r$, cioè $p + 1$, ponendo $p = 2m + r + 1$, sono in relazione nel seguente modo:

$$p = n + m + 1$$

3. **Interpolazione dei punti estremi:** $C(0) = P_0$, $C(1) = P_n$;

4. **Invarianza affine:** Applicare una trasformazione affine ad una curva B-spline è equivalente ad applicare la stessa ai suoi punti di controllo;

5. **Proprietà di guscio strettamente convesso:** La curva è contenuta nel guscio convesso della sua poligonale di controllo; infatti, se $u \in [u_i, u_{i+1})$, $m \leq i < p - m - 1 = n$, $C(u)$ è nel guscio convesso dei punti di controllo $P_{i-m}, ..., P_i$;

6. **Metodo di modificazione locale:** muovendo $P_i$, cambia $C(u)$ solo nell'intervallo $[u_i, u_{i+m+1})$;

7. Il poligono di controllo rappresenta un'approssimazione della curva; questa approssimazione può essere migliorata con l'inserzione di nodi aggiuntivi o elevando il grado;

8. Muovendosi lungo la curva da $u = 0$ ad $u = 1$, le funzioni $N_{i,m}(u)$ agiscono come interruttori: nel momento in cui $u$ attraversa un nodo, una $N_{i,m}(u)$ (e il corrispondente $P_i$) si spegne e la successiva si accende;

9. **Proprietà di variation diminishing:** Nessuna retta del piano ha più intersezioni con la curva rispetto a quante ne abbia con la poligonale di controllo associata;

10. $C(u)$ risulta infinitamente differenziabile all'interno degli intervalli nodali e differenziabile almeno $m - m_i$ volte in corrispondenza di un nodo di molteplicità $m_i$.

Cf. [PT97] per maggiori dettagli.

### 1.3.4 Alcuni algoritmi geometrici per le spline

#### Inserzione nodale

Sia $C(u) = N_{i,m}(u)P_i$ una curva B-spline definita su $\Delta^* = \{u_0, ..., u_p\}$. Sia $\hat{u} \in [u_k, u_{k+1})$; inseriamo $\hat{u}$ in $\Delta^*$ in modo da formare un nuovo vettore no-

dale $\hat{\Delta}^* = \{\hat{u}_0 = u_0, ..., \hat{u}_k = u_k, \hat{u}_{k+1} = \hat{u}, \hat{u}_{k+2} = u_{k+1}, ..., \hat{u}_{p+1} = u_p\}$.
Se $S(P_m, M, \Delta)$ e $S(P_m, \hat{M}, \hat{\Delta})$ denotano gli spazi vettoriali delle curve definite su $\Delta^*$ e $\hat{\Delta}^*$, rispettivamente, allora $S(P_m, M, \Delta) \subset S(P_m, \hat{M}, \hat{\Delta})$; inserendo un nodo, infatti, aumenta la dimensione dello spazio, e

$$C(u) \in S(P_m, M, \Delta) \Rightarrow C(u) \in S(P_m, \hat{M}, \hat{\Delta})$$

Se le $N_{i,m}(u)$, $i = 0, ..., n$ sono una base per $S(P_m, M, \Delta)$ e le $\hat{N}_{i,m}(u)$, $i = 0, ..., n+1$ sono una base per $S(P_m, \hat{M}, \hat{\Delta})$, si vuole trovare la rappresentazione di $C(u)$ nella nuova base conoscendone la rappresentazione nella vecchia. Un metodo potrebbe essere quello di risolvere un problema di interpolazione, ma ciò comporterebbe la risoluzione di un sistema lineare. Un metodo migliore ci viene dato dal seguente

**Teorema 1.7** Siano $\Delta^*$ e $\hat{\Delta}^*$ le due partizioni nodali definite, allora vale la seguente relazione:

$$N_{i,m}(u) = \begin{cases} \hat{N}_{i,m}(u) & i \leq k - m \\ \frac{\hat{u}-\hat{u}_i}{\hat{u}_{i+m}-\hat{u}_i}\hat{N}_{i,m}(u) + \frac{\hat{u}_{i+m+1}-\hat{u}}{\hat{u}_{i+m+1}-\hat{u}_{i+1}}\hat{N}_{i+1,m}(u) & k - m - 1 \leq i \leq k \\ \hat{N}_{i+1,m}(u) & i \geq k + 1 \end{cases}$$

**Corollario 1.2** Siano $\Delta^*$ e $\hat{\Delta}^*$ le due partizioni nodali prima definite, allora:

$$Q_i = \begin{cases} P_i & i \leq k - m + 1 \\ \lambda_i P_i + (1 - \lambda_i)P_{i-1} & k - m + 2 \leq i \leq k \\ P_{i-1} & i \geq k + 1 \end{cases}$$

dove $\lambda_i = \frac{\hat{u}-\hat{u}_i}{\hat{u}_{i+m}-\hat{u}_i}$, i $P_i$ sono i coefficienti nella vecchia base e i $Q_i$ sono i coefficienti nella nuova base.

**Osservazione 1.6** I $\lambda_i$, per come sono definiti, sono tutti numeri in $[0, 1] \, \forall \, i$, quindi, i coefficienti $Q_i$ sono una combinazione convessa dei $P_i$.

È importante notare che l'inserzione nodale rappresenta solamente un

cambiamento della base dello spazio vettoriale, quindi la curva non subisce variazioni geometriche (il supporto resta lo stesso).

### Valutazione della spline in un punto

Per calcolare il valore di una curva spline di grado $m$ in un punto del dominio parametrico, si può adattare l'algoritmo di deCasteljau, utilizzato per le curve di Bézier, tenendo conto che il dominio parametrico è suddiviso in intervalli nodali.
Sia $u \in [u_j, u_{j+1})$. Si ha:

$$\begin{aligned}
C(u) &= \sum_{i=j-m}^{j} P_i N_{i,m}(u) \\
&= \sum_{i=j-m}^{j} P_i \left[ \frac{u-u_i}{u_{i+m}-u_i} N_{i,m-1}(u) + \frac{u_{i+m+1}-u}{u_{i+m+1}-u_{i+1}} N_{i+1,m-1}(u) \right] \\
&= \sum_{i=j-m+1}^{j} P_i \frac{u-u_i}{u_{i+m}-u_i} N_{i,m-1}(u) + \sum_{i=j-m}^{j-1} P_i \frac{u_{i+m+1}-u}{u_{i+m+1}-u_{i+1}} N_{i+1,m-1}(u) \\
&= \sum_{i=j-m+1}^{j} \frac{P_i(u-u_i) + P_{i-1}(u_{i+m}-u)}{u_{i+m}-u_i} N_{i,m-1}(u)
\end{aligned}$$

Poniamo $P_i^{[1]} = \frac{P_i(u-u_i)+P_{i-1}(u_{i+m}-u)}{u_{i+m}-u_i}$

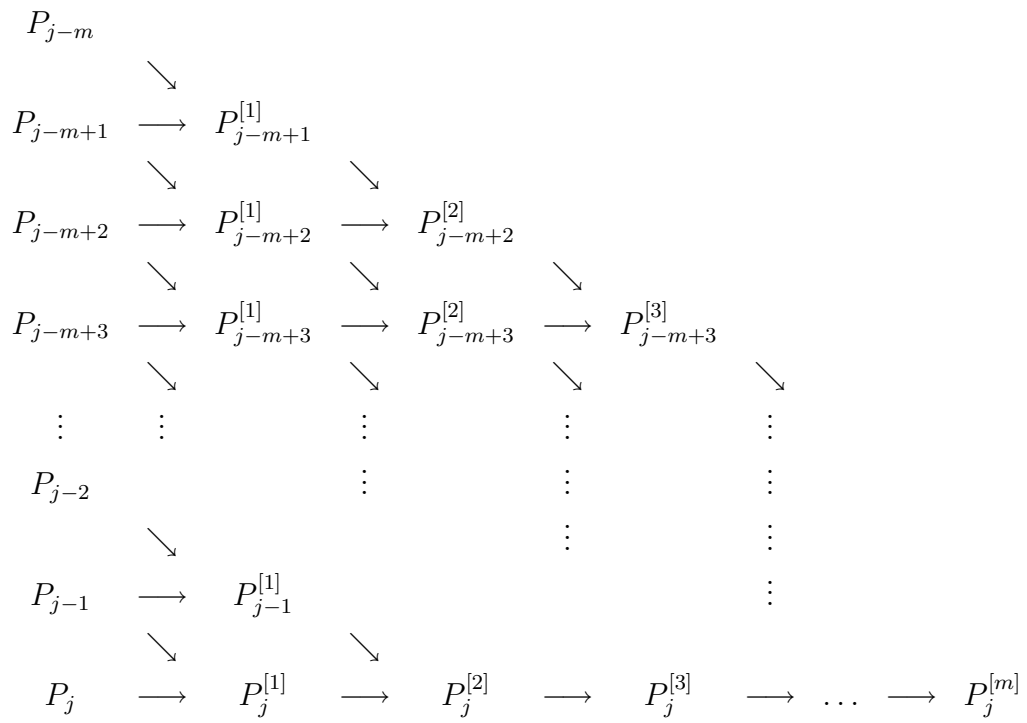
$$\begin{aligned}
C(u) &= \sum_{i=j-m+1}^{j} P_i^{[1]} N_{i,m-1}(u) \\
&= \sum_{i=j-m+2}^{j} P_i^{[1]} \frac{u-u_i}{u_{i+m-1}-u_i} N_{i,m-2}(u) + \sum_{i=j-m+1}^{j-1} P_i^{[1]} \frac{u_{i+m}-u}{u_{i+m}-u_{i+1}} N_{i+1,m-2}(u) \\
&= \sum_{i=j-m+2}^{j} \frac{P_i^{[1]}(u-u_i) + P_{i-1}^{[1]}(u_{i+m-1}-u)}{u_{i+m-1}-u_i} N_{i,m-2}(u)
\end{aligned}$$

Poniamo $P_i^{[2]} = \frac{P_i^{[1]}(u-u_i)+P_{i-1}^{[1]}(u_{i+m-1}-u)}{u_{i+m-1}-u_i}$

$$= \sum_{i=j-m+2}^{j} P_i^{[2]} N_{i,m-2}(u)$$

$$= \sum_{i=j-m+3}^{j} P_i^{[3]} N_{i,m-3}(u) = \ldots = \sum_{i=j-m+m}^{j} P_i^{[m]} N_{i,m-m}(u) = P_j^{[m]}$$

Tramite uno schema ad albero, questo algoritmo può essere rappresentato nel modo seguente:




Cf. [eGA98] per maggiori dettagli.

# Capitolo 2

# Funzione di Taglia di una Spline

Nella prima parte di questo capitolo analizzeremo curve spline di secondo grado definite su partizioni nodali uniformi (i nodi, cioè, sono equidistanti due a due) a nodi aggiuntivi coincidenti. In particolare, il nostro obiettivo consiste nell'individuare la posizione di un punto critico di una tale curva, rispetto ad una particolare funzione misurante, indipendentemente dal numero di punti di controllo, ma derivante solamente dalle caratteristiche locali della curva. La funzione misurante scelta è la funzione altezza, cioè la funzione che ad ogni punto della curva nel piano associa la sua ordinata.
Nella seconda parte, invece, ci concentreremo sul problema di rappresentazione delle funzioni di taglia associate alle curve spline.

## 2.1 Punti critici di una curva spline

**Lemma 2.1** Sia $C(u)$ una curva B-spline parametrica di secondo grado, definita per ogni $u \in [0,1]$, con partizione nodale uniforme, e sia $\{P_i\}_{i=0,...,n}$ la successione di punti di controllo associata. Allora:

$$\frac{P_i + P_{i+1}}{2} \in C(u), \quad i = 1, ..., n-2$$

*Dimostrazione.* Poiché la partizione nodale è uniforme, ogni nodo interno all'intervallo $(0,1)$ è esprimibile nella forma: $u_{i+m} = \frac{i}{n-m+1}$, dove $m$ è il grado della curva (quindi, nel nostro caso, è pari a 2), $i \in \{1, ..., n-m\}$ e $n+1$ è il numero di punti di controllo.
Studiando la base di funzioni B-spline normalizzate, osserviamo che, per la proprietà di supporto locale, in corrispondenza del nodo $u_{i+2}$, $\forall\, i$, solamente le seguenti funzioni risultano non nulle: $N_{i,2}(u), N_{i+1,2}(u)$; inoltre, per la proprietà di partizione dell'unità, $N_{i,2}(u_{i+2}) + N_{i+1,2}(u_{i+2}) = 1$. Allora:

$$\begin{aligned}
C(u_{i+2}) &= N_{i,2}(u_{i+2})P_i + N_{i+1,2}(u_{i+2})P_{i+1} \\
&= \left[\frac{u_{i+2}-u_i}{u_{i+2}-u_i}N_{i,1}(u_{i+2}) + \frac{u_{i+3}-u_{i+2}}{u_{i+3}-u_{i+1}}N_{i+1,1}(u_{i+2})\right] P_i \\
&+ \left[\frac{u_{i+2}-u_{i+1}}{u_{i+3}-u_{i+1}}N_{i+1,1}(u_{i+2}) + \frac{u_{i+4}-u_{i+2}}{u_{i+4}-u_{i+2}}N_{i+2,1}(u_{i+2})\right] P_{i+1} \\
&= \left[N_{i,1}(u_{i+2}) + \frac{1}{2}N_{i+1,1}(u_{i+2})\right] P_i + \left[\frac{1}{2}N_{i+1,1}(u_{i+2}) + N_{i+2,1}(u_{i+2})\right] P_{i+1} \\
&= \left[\frac{u_{i+2}-u_i}{u_{i+1}-u_i}N_{i,0}(u_{i+2}) + \frac{u_{i+2}-u_{i+2}}{u_{i+2}-u_{i+1}}N_{i+1,0}(u_{i+2})\right] P_i \\
&+ \frac{1}{2}\left[\frac{u_{i+2}-u_{i+1}}{u_{i+2}-u_{i+1}}N_{i+1,0}(u_{i+2}) + \frac{u_{i+3}-u_{i+2}}{u_{i+3}-u_{i+2}}N_{i+2,0}(u_{i+2})\right] (P_i + P_{i+1}) \\
&= \frac{1}{2}N_{i+2,0}(u_{i+2})(P_i + P_{i+1}) \\
&= \frac{1}{2}(P_i + P_{i+1})
\end{aligned}$$

cioè $N_{i,2}(u_{i+2}) = N_{i+1,2}(u_{i+2}) = \frac{1}{2}$. □

Il primo passo che vogliamo compiere è quello di rappresentare la spline come unione di curve definite ognuna su un intervallo nodale del dominio parametrico di $C(u)$, $[0,1]$. In questo modo, è possibile studiare ciascun punto critico, rispetto alla funzione misurante scelta, localmente, disinteressandosi, quindi, dell'andamento globale della curva.
Applichiamo, dunque, a $C(u)$ l'algoritmo d'inserzione nodale nel seguente modo: in corrispondenza di ogni punto $u_{i+2} \in (0,1)$, inseriamo 2 nodi aggiuntivi coincidenti. Così facendo, la curva $C(u)$ non subisce alcuna variazione dal punto di vista geometrico. Ciò che cambia è che ora può essere

studiata come unione di curve di Bézier di secondo grado; infatti, nel dominio parametrico, le funzioni base B-spline risultano a 3 a 3 non nulle sugli intervalli nodali $[u_{i+1}, u_{i+2}]$, $i = 1, ..., n-1$ e coincidono con i polinomi di Bernstein.
Ora, il loro numero non è più $n+1$, bensì $3(n-1)$; infatti, se $p+1$, cioè il numero totale dei nodi, era pari a $m+1+n+1 = n+4$, ora è uguale a $3(n-2)+2(m+1) = 3n-6+6 = 3n$, e quindi, affinchè resti valida la relazione tra grado, numero di nodi e numero di B-spline non nulle, se con $x$ indichiamo quest'ultimo, si ha: $3n = 3+x \Rightarrow x = 3n-3$. Quindi anche il numero di punti di controllo diventa pari a $3n-3$, ed in particolare, dovremo aggiungere ai precedenti $\{P_i\}_{i=0,...,n}$ due punti di controllo coincidenti in corrispondenza di ogni punto medio $\frac{P_i+P_{i+1}}{2}$, $i = 1, ..., n-2$.

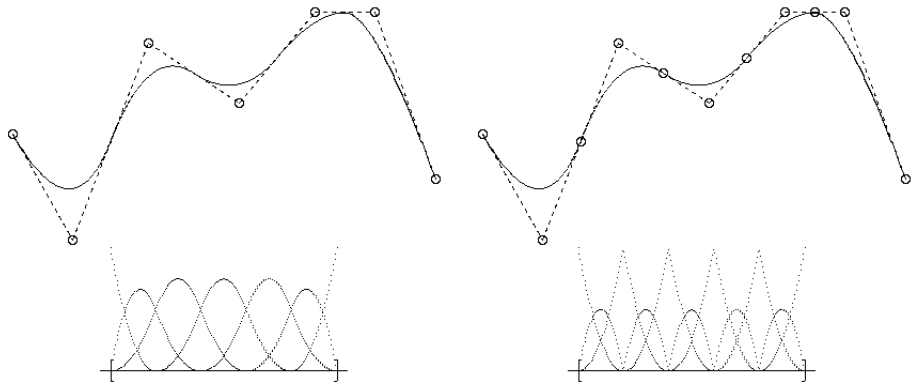


Figura 2.1: Esempio di applicazione dell'algoritmo di knot-refinement. Il supporto delle due curve è lo stesso; ciò che cambia è il numero delle funzioni base.

Possiamo così definire una nuova successione di punti associata a $C(u)$.

**Lemma 2.2** Sia $\{Q_j\}_{j=0,...,3n-4}$ la successione di punti associata a $C(u)$, ottenuta da $\{P_i\}_{i=0,...,n}$ applicando alla curva l'algoritmo d'inserzione nodale,

in modo tale che:

$$\begin{aligned} Q_0 &= P_0 \\ Q_{3j+1} &= P_{j+1}, \qquad j = 0, ..., n-2 \\ Q_{3j+2} \equiv Q_{3j+3} &= \tfrac{P_{j+1}+P_{j+2}}{2}, \quad j = 0, ..., n-3 \\ Q_{3n-4} &= P_n \end{aligned}$$

Allora vale la seguente espressione:

$$\sum_{i=0}^{n} N_{i,2}(u)P_i = \sum_{k=0}^{2} B_{k,2}(v) \sum_{i=0}^{n-2} Q_{3i+k}$$

dove $u \in [0,1]$, $v \in [u_{i+2}, u_{i+3}], i = 0, ..., n-2$ e

$$\begin{aligned} B_{0,2}(v) &= \frac{(u_{i+3}-v)^2}{(u_{i+3}-u_{i+2})^2} \\ B_{1,2}(v) &= 2\frac{(v-u_{i+2})(u_{i+3}-v)}{(u_{i+3}-u_{i+2})^2} \\ B_{2,2}(v) &= \frac{(v-u_{i+2})^2}{(u_{i+3}-u_{i+2})^2} \end{aligned}$$

sono i polinomi base di Bernstein definiti sugli intervalli nodali $[u_{i+2}, u_{i+3}]$, $i = 0, ..., n-2$

*Dimostrazione.* Sia $u \in [u_{i+2}, u_{i+3}] \subset [0,1]$. Per la proprietà di supporto locale, le uniche B-spline non nulle sono: $N_{i,2}(u), N_{i+1,2}(u), N_{i+2,2}(u)$, e quindi

$$C(u) = \sum_{i=0}^{n} N_{i,2}(u)P_i = N_{i,2}(u)P_i + N_{i+1,2}(u)P_{i+1} + N_{i+2,2}(u)P_{i+2}, \; u \in [u_{i+2}, u_{i+3}]$$

e, su tale intervallo, queste funzioni sono pari a

$$\begin{cases} N_{i,2}(u) &= \frac{u_{i+3}-u}{u_{i+3}-u_{i+1}} \frac{u_{i+3}-u}{u_{i+3}-u_{i+2}} \\ N_{i+1,2}(u) &= \frac{u-u_{i+1}}{u_{i+3}-u_{i+1}} \frac{u_{i+3}-u}{u_{i+3}-u_{i+2}} + \frac{u_{i+4}-u}{u_{i+4}-u_{i+2}} \frac{u-u_{i+2}}{u_{i+3}-u_{i+2}} \\ N_{i+2,2}(u) &= \frac{u-u_{i+2}}{u_{i+4}-u_{i+2}} \frac{u-u_{i+2}}{u_{i+3}-u_{i+2}} \end{cases}$$

Allora l'equazione parametrica di questo tratto di curva diventa:

$$\begin{aligned} C(u) &= \frac{(u_{i+3}-u)^2}{(u_{i+3}-u_{i+1})(u_{i+3}-u_{i+2})}P_i \\ &+ \left[\frac{(u-u_{i+1})(u_{i+3}-u)}{(u_{i+3}-u_{i+1})(u_{i+3}-u_{i+2})} + \frac{(u_{i+4}-u)(u-u_{i+2})}{(u_{i+4}-u_{i+2})(u_{i+3}-u_{i+2})}\right]P_{i+1} \\ &+ \frac{(u-u_{i+2})^2}{(u_{i+4}-u_{i+2})(u_{i+3}-u_{i+2})}P_{i+2} \end{aligned}$$

A questo punto usiamo la regolarità della partizione nodale: i nodi interni sono semplici ed equidistanti tra loro, quindi:

$$\begin{aligned} &u_{i+3}-u_{i+1} = u_{i+4}-u_{i+2} = 2(u_{i+3}-u_{i+2}) \\ &u-u_{i+1} = (u-u_{i+2})+(u_{i+2}-u_{i+1}) = (u-u_{i+2})+(u_{i+3}-u_{i+2}) \\ &u_{i+4}-u = (u_{i+4}-u_{i+3})+(u_{i+3}-u) = (u_{i+3}-u_{i+2})+(u_{i+3}-u) \end{aligned}$$

Facendo le dovute sostituzioni nell'espressione di $C(u)$, $u \in [u_{i+2}, u_{i+3}]$, otteniamo:

$$\begin{aligned} C(u) &= \frac{(u_{i+3}-u)^2}{(u_{i+3}-u_{i+2})^2}\frac{P_i+P_{i+1}}{2} + \frac{(u-u_{i+2})(u_{i+3}-u)}{(u_{i+3}-u_{i+2})^2}P_{i+1} \\ &+ \frac{(u-u_{i+2})^2}{(u_{i+3}-u_{i+2})^2}\frac{P_{i+1}+P_{i+2}}{2} \\ &= B_{0,2}(v)Q_{3i} + B_{1,2}(v)Q_{3i+1} + B_{2,2}(v)Q_{3i+2}, \quad v \in [u_{i+2}, u_{i+3}] \end{aligned}$$

□

Prima di concentrarci sullo studio dei punti critici di $\mathcal{M}$ rispetto a $\varphi$, effettuiamo una trasformazione del dominio parametrico del tipo:

$$f : [a,b] \longrightarrow [c,d], \quad f(w) = w' := \frac{w-a}{b-a}(d-c)+c$$

con cui vogliamo mappare l'intervallo $[u_{i+2}, u_{i+3}]$ in $[0,1]$ e studiare su questo le B-spline come curve di Bézier. Un polinomio di Bernstein, infatti, risulta invariante rispetto a questo tipo di trasformazione. Verifichiamolo.

**Lemma 2.3** Sia

$$f : [a,b] \longrightarrow [c,d], \quad f(w) = w' := \frac{w-a}{b-a}(d-c)+c$$

Allora il seguente diagramma

$$\begin{array}{ccc} [a,b] & \xrightarrow{f} & [c,d] \\ B_{i,n}(w) \searrow & & \swarrow B_{i,n}(w') \\ & [0,1] & \end{array}$$

è commutativo, cioè

$$B_{i,n}(w) = B_{i,n}(f(w)), \quad f(w) = w'$$

.

*Dimostrazione.*

$$\begin{aligned} B_{i,n}(w') &= \binom{n}{i} \frac{(w'-c)^i (d-w')^{n-i}}{(d-c)^n} \\ &= \binom{n}{i} \frac{1}{(d-c)^n} \frac{(w-a)^i (d-c)^i}{(b-a)^i} \left[ d - \frac{w-a}{b-a}(d-c) - c \right]^{n-i} \\ &= \binom{n}{i} \frac{(w-a)^i (d-c)^i}{(b-a)^n (d-c)^n} (d-c)^{n-i} (b-w)^{n-i} \\ &= \binom{n}{i} \frac{(w-a)^i (b-w)^{n-i}}{(b-a)^n} = B_{i,n}(w) \end{aligned}$$

□

Applicando $f$ a $B_{0,2}(v), B_{1,2}(v), B_{2,2}(v)$, con $v \in [u_{j+2}, u_{j+3}]$, otteniamo:

$$\begin{aligned} B_{0,2}(u) &= (1-u)^2 \\ B_{1,2}(u) &= 2u(1-u) \\ B_{2,2}(u) &= u^2 \end{aligned}$$

con $u \in [0,1]$.

Finalmente siamo giunti al risultato principale di questo capitolo; il seguente teorema, infatti, ci dice dove si trovano i punti critici, rispetto alla funzione altezza, nel grafico di una spline di secondo grado, con partizione nodale uniforme in relazione alla sua poligonale di controllo.

**Teorema 2.1** Sia $(\mathcal{M}, \varphi)$ una coppia di taglia, dove $\mathcal{M}$ è il grafico di una curva spline $C(u)$ di secondo grado, con punti di controllo distinti, definita sul dominio parametrico $[0, 1]$, con partizione nodale uniforme, e $\varphi$ è la funzione misurante definita nel seguente modo: $\varphi : \mathcal{M} \longrightarrow \mathbb{R}$, $\varphi(x, y) = y$. Allora ogni punto critico di $\mathcal{M}$ rispetto a $\varphi$ si trova o nel punto medio di un segmento della poligonale di controllo, oppure su una parabola della forma:

$$x = Ay^2 + By + C$$

*Dimostrazione.* Chiamiamo $Q_k \equiv Q_{k+1} = \frac{1}{2}(P_i + P_{i+1}) = (x_k, y_k)$ e distinguiamo i seguenti due casi:

1. Esistono due punti di controllo consecutivi, $P_i \equiv Q_{k-1} = (x_{k-1}, y_{k-1})$, $P_{i+1} \equiv Q_{k+2} = (x_{k+2}, y_{k+2})$ con $y_{k-1} = y_{k+2}$, cioè alla stessa altezza. Allora $y_{k-1} = y_k = y_{k+2}$. Consideriamo tre sottocasi:

   1.i Se $Q_{k-2}$ e $Q_{k+3}$ sono entrambi nel semipiano inferiore rispetto alla retta per $P_i, P_{i+1}$, cioè $y_{k-2} < y_k$, $y_{k+3} < y_k$, allora $Q_k$ è un punto di massimo locale per $\varphi$;

   1.ii Se $Q_{k-2}$ e $Q_{k+3}$ sono entrambi nel semipiano superiore rispetto alla retta per $P_i, P_{i+1}$, cioè $y_{k-2} > y_k$, $y_{k+3} > y_k$, allora $Q_k$ è un punto di minimo locale per $\varphi$;

   1.iii Se $Q_{k-2}$ e $Q_{k+3}$ sono in semipiani diversi rispetto alla retta per $P_i, P_{i+1}$, allora $Q_k$ è un punto di flesso a tangente orizzontale. In questo tratto la curva risulta monotona e quindi non ci sono punti critici per $\varphi$.

   Se $P_i, ..., P_{i+h}, h \geq 2$ sono allineati su un segmento parallelo all'asse delle ascisse e $P_{i-1}, P_{i+h+1}$ sono nello stesso semipiano rispetto alla retta da essi generata, allora $\overline{\frac{P_i+P_{i+1}}{2} \frac{P_{i+h-1}+P_{i+h}}{2}}$ costituisce un segmento di punti critici per $\varphi$.

2. Esistono due punti di controllo consecutivi, $P_i \equiv Q_{k-1} = (x_{k-1}, y_{k-1})$,

$P_{i+1} \equiv Q_{k+2} = (x_{k+2}, y_{k+2})$ con $y_{k-1} \neq y_{k+2}$.
Supponiamo $y_{k-1} > y_{k+2}$, da cui, $y_{k-1} > y_k > y_{k+2}$. Allora:

2.i Se $y_{k-2} > y_{k-1}$ (analogamente, $y_{k+3} < y_{k+2}$), nel tratto di curva $Q_{k-2}, Q_{k-1}, Q_k$ (analogamente, $Q_k, Q_{k+2}, Q_{k+3}$), la curva risulta monotona decrescente, e quindi $\varphi$ non ammette punti critici su questo tratto;

2.ii Se $y_{k-2} = y_{k-1}$ (analogamente, $y_{k+3} = y_{k+2}$), si torna al caso 1. e, in particolare, ad uno dei tre sottocasi, a seconda della posizione di $Q_{k-3}$ (rispettivamente di $Q_{k+4}$);

2.iii Se $y_{k-2} < y_{k-1}$ (analogamente, $y_{k+3} > y_{k+2}$), allora esiste un punto di massimo (rispettivamente di minimo)per $\varphi$ e si trova su una parabola con asse parallelo all'asse delle ascisse.

Dimostriamo il caso 2.iii supponendo che siano verificate le ipotesi richieste sul tratto di spline determinato dai punti $Q_{k-2}, Q_{k-1}, Q_k$.
Per la proprietà di invarianza affine di cui godono queste funzioni, per semplificare i calcoli, possiamo porre uno dei tre punti nell'origine, ad esempio, prendiamo $Q_{k-1} = (0, 0)$. Ora, supponiamo che i due punti estremi siano alla stessa altezza, cioè poniamo $Q_{k-2} = (a, b), Q_k = (c, b)$, con $b \neq 0$. Allora questo tratto di curva, essendo, in particolare, una curva di Bézier, può essere espresso nella forma:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} (1-u)^2 + \begin{pmatrix} 0 \\ 0 \end{pmatrix} 2u(1-u) + \begin{pmatrix} c \\ b \end{pmatrix} u^2 = \begin{pmatrix} a - 2au + (a+c)u^2 \\ b - 2bu + 2bu^2 \end{pmatrix}$$

moltiplichiamo ognuna delle due equazioni per il coefficiente del termine di secondo grado dell'altra

$$\begin{cases} 2bx &= 2ab - 4abu + 2b(a+c)u^2 \\ (a+c)y &= (a+c)b - 2b(a+c)u + 2b(a+c)u^2 \end{cases}$$

e sottraiamo membro a membro, così da avere il parametro $u$ in forma lineare:

$$\Rightarrow\ u = \frac{2bx - (a+c) - b(a+c)}{2bc}, \qquad u \in [0, 1]$$

Sostituiamo il valore trovato in una delle equazioni a sistema per ottenere l'equazione cartesiana di una parabola:

$$4b^2(a+c)x^2 + (a+c)^3y^2 - 4b(a+c)^2xy + \\ -2b(a^2-c^2)(a-c)y + b^2(a^2-c^2)(a-c) = 0$$

Vogliamo calcolarne il punto critico rispetto alla funzione altezza $\varphi$.
Mettiamo quindi a sistema l'equazione sopra con una retta parallela all'asse delle ascisse e imponiamo che la loro intersezione sia costituita da due punti coincidenti, cioè richiediamo che il delta sia nullo.

$$\begin{cases} 4b^2(a+c)x^2 + (a+c)^3y^2 - 4b(a+c)^2xy + \\ -2b(a^2-c^2)(a-c)y + b^2(a^2-c^2)(a-c) = 0 \\ y = h,\ h \in \mathbb{R} \end{cases}$$

$$\Rightarrow\ 4b^2(a+c)x^2 - 4b(a+c)^2hx + (a+c)^3h^2 + b(a^2-c^2)(a-c)(b-2h) = 0$$

Allora $\frac{\Delta}{4} = 0$ se e solo se:

$$[2b(a+c)^2h]^2 - 4b^2(a+c)[(a+c)^3h^2 + b(a^2-c^2)(a-c)(b-2h)] = 0$$
$$\Leftrightarrow\ 4b^3(a^2-c^2)(2h-b) = 0$$

da cui

$$y = h = \frac{b}{2}$$

Andando a sostituire nella formula ridotta di risoluzione dell'equazione di secondo grado, otteniamo:

$$x = x_{1,2} = \frac{2b(a+c)^2}{4b^2(a+c)}\frac{b}{2}$$

$$\Rightarrow \qquad x = \frac{a+c}{4}$$

Risulta quindi che il punto critico della parabola rispetto a $\varphi$ ha le seguenti coordinate:

$$V = \left(\frac{a+c}{4}, \frac{b}{2}\right)$$

cioè è il punto medio del segmento $\overline{Q_{k-1}M}$, dove $M = \frac{Q_{k-2}+Q_k}{2}$.
Ricaviamo ora l'equazione cartesiana dalla parabola passante per $Q_{k-1}, V, Q_k$ e con asse parallelo all'asse delle ascisse:

$$x = \frac{c-a}{b^2}y^2 + \frac{a}{b}y \tag{2.1}$$

Ora, vogliamo provare che, sostituendo $Q_{k-2}$ con $Q'_{k-2}$, punto generico appartenente alla semiretta per $Q_{k-2}, Q_{k-1}$ (di equazione $y = \frac{b}{a}x$, tale per cui, se $b < 0$, anche $\frac{b}{a}x \leq 0$, mentre se $b > 0$, anche $\frac{b}{a}x \geq 0$), il punto critico rispetto a $\varphi$ della nuova curva per $Q'_{k-2} = (d, \frac{bd}{a}), Q_{k-1} = (0,0), Q_k = (c,b)$ giace sempre su questa parabola.
Consideriamo:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} d \\ \frac{bd}{a} \end{pmatrix}(1-u)^2 + \begin{pmatrix} 0 \\ 0 \end{pmatrix}2u(1-u) + \begin{pmatrix} c \\ b \end{pmatrix}u^2$$

$$= \begin{pmatrix} d - 2du + (c+d)u^2 \\ \frac{bd}{a} - 2\frac{bd}{a}u + 2b\left(\frac{d}{a}+1\right)u^2 \end{pmatrix}$$

Procedendo come prima, otteniamo:

$$\begin{cases} 2b\left(\frac{d}{a}+1\right)x &= 2b\left(\frac{d}{a}+1\right)d - 22b\left(\frac{d}{a}+1\right)du + 2b\left(\frac{d}{a}+1\right)(c+d)u^2 \\ (c+d)y &= \frac{bd}{a}(c+d) - 2\frac{bd}{a}(c+d)u + 2b\left(\frac{d}{a}+1\right)(c+d)u^2 \end{cases}$$

$$\Rightarrow u = \frac{-b(d+a)x + a(c+d)y + b(a-c)d}{2b(a-c)d}$$

Sostituendo $u$ in una delle due equazioni a sistema, otteniamo una famiglia (parametrizzata mediante $d$) di parabole passanti per $Q_k$, con la seguente equazione cartesiana:

$$\begin{aligned} &b^2(a+d)^2(c+d)x^2 + a^2(c+d)^3y^2 + \\ &-2ab(a+d)(c+d)^2xy - 2b^2d(a-c)(c+d)(a-d)x + \\ &+2abd(a-c)(c^2-d^2)y + b^2d^2(a-c)^2(c+d) = 0 \end{aligned}$$

Imponiamo come prima la tangenza ad una retta parallela all'asse delle ascisse, mettendo a sistema

$$\begin{cases} b^2(a+d)^2(c+d)x^2 + a^2(c+d)^3y^2 + \\ -2ab(a+d)(c+d)^2xy - 2b^2d(a-c)(c+d)(a-d)x + \\ +2abd(a-c)(c^2-d^2)y + b^2d^2(a-c)^2(c+d) = 0 \\ y = h,\ h \in \mathbb{R} \end{cases}$$

$$\Rightarrow \quad b^2(a+d)^2(c+d)x^2 - 2b(c+d)[a(a+d)(c+d)h + bd(a-c)(a-d)]x + \\ +a^2(c+d)^3h^2 + 2abd(a-c)(c^2-d^2)h + b^2d^2(a-c)^2(c+d) = 0$$

e richiedendo $\frac{\Delta}{4} = 0$:

$$b^2(c+d)^2[a(a+d)(c+d)h + bd(a-c)(a-d)]^2 + \\ -b^2(a+d)^2(c+d)[a^2(c+d)^3h^2 + 2abd(a-c)(c^2-d^2)h + \\ +b^2d^2(a-c)^2(c+d)] = 0$$

$$\Leftrightarrow \quad 4ab^3d^2(a-c)^2(a+d)(c+d)^2h = 4ab^4d^3(a-c)^2(c+d)^2$$

Semplificando:

$$(a+d)h = bd$$

$$\Rightarrow \quad y = h = \frac{bd}{a+d}$$

Allora:

$$x = x_{1,2} = \frac{b(c+d)[a(a+d)(c+d)\frac{bd}{a+d} + bd(a-c)(a-d)]}{b^2(a+d)^2(c+d)}$$

$$\Rightarrow \quad x = \frac{d[a(c+d) + (a-c)(a-d)]}{(a+d)^2}$$

$$\Rightarrow \quad x = \frac{d(a^2+cd)}{(a+d)^2}$$

Chiamiamo

$$V' = \left( \frac{d(a^2 + cd)}{(a+d)^2}, \frac{bd}{a+d} \right), \quad \forall\, d$$

il punto critico rispetto a $\varphi$ e inseriamone le coordinate nell'equazione (2.1); si può facilmente verificare la sua appartenenza a tale parabola.
A questo punto, per la proprietà di invarianza affine, basta effettuare una traslazione sui punti di controllo per giungere all'equazione di una parabola generica: $x = Ay^2 + By + C$. □

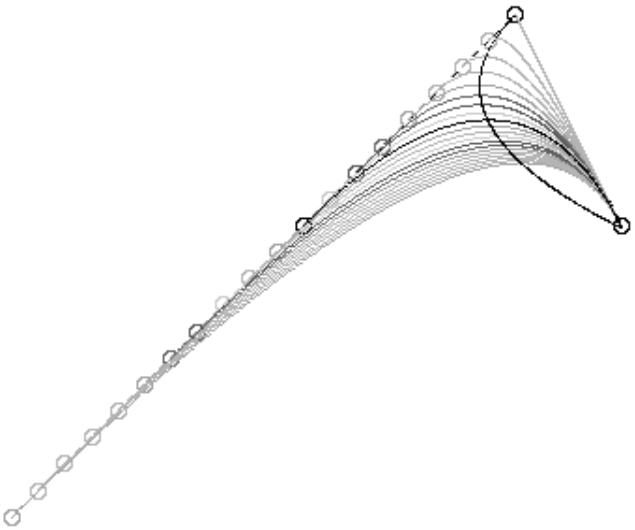


Figura 2.2: Parabola dei punti critici relativa ad una famiglia di curve spline.

**Osservazione 2.1** Naturalmente, se avessimo tenuto fisso il punto $Q_{k-2} = (a, b)$ e avessimo sostituito $Q_k = (c, b)$ con $Q'_k$, punto generico appartenente alla semiretta di origine $Q_{k-1}$ e passante per $Q_k$, di equazione $y = \frac{b}{c}x$, il punto critico rispetto a $\varphi$ lo avremmo trovato sulla parabola per $Q_{k-2} = (a, b), V, Q_{k-1} = (0, 0)$, che ha ancora asse parallelo all'asse delle ascisse, ma concavità rivolta nel verso opposto a quella precedentemente considerata.

**Definizione 2.1** Sia $(\mathcal{M}, \varphi)$ una coppia di taglia, dove $\mathcal{M}$ è il grafico di

una curva spline di secondo grado, definita sul dominio parametrico $[0,1]$, con partizione nodale uniforme, e $\varphi$ è la funzione misurante definita nel seguente modo: $\varphi : \mathcal{M} \longrightarrow \mathbb{R}$, $\varphi(x,y) = y$. Definiamo *parabola dei punti critici rispetto alla funzione altezza* la curva di equazione:

$$x = Ay^2 + By + C$$

**Teorema 2.2** Siano dati tre punti $Q_{k-2} = (x_{k-2}, y_{k-2})$, $Q_{k-1} = (x_{k-1}, y_{k-1})$, $Q_k = (x_k, y_k)$ distinti sulla poligonale di controllo $\mathcal{P}$, che verifichino la condizione 2.iii, e la curva di Bézier associata. Allora, sostituendo $Q_{k-2}$ con $Q'_{k-2}$, punto arbitrario che varia lungo la semiretta di origine $Q_{k-1}$ e passante per $Q_{k-2}$, la parabola dei punti critici rispetto a $\varphi$ coincide con una curva di Bézier di secondo grado i cui punti di controllo sono:

$$Q_{k-1}, \frac{\overline{QQ_{k-1}}}{2}, Q_k, \quad \text{con} \quad Q = \left( \frac{x_{k-2} - x_{k-1}}{y_{k-2} - y_{k-1}}(y_k - y_{k-1}) + x_{k-1}, y_k \right).$$

*Dimostrazione.* Siano $Q_{k-2} = (a,b), Q_{k-1} = (0,0), Q_k = (c,b), b \neq 0$. Sostituiamo $Q_{k-2}$ con $Q'_{k-2} = (d, \frac{bd}{a})$, con parametro $d$, come prima. Allora possiamo verificare che $\frac{\overline{QQ_{k-1}}}{2} = \left(\frac{a}{2}, \frac{b}{2}\right)$.
Ora, la curva di Bézier associata a $Q_{k-1}, \frac{\overline{QQ_{k-1}}}{2}, Q_k$ è descritta dalla seguente equazione:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}(1-u)^2 + \begin{pmatrix} \frac{a}{2} \\ \frac{b}{2} \end{pmatrix} 2u(1-u) + \begin{pmatrix} c \\ b \end{pmatrix} u^2 = \begin{pmatrix} au + (c-a)u^2 \\ bu \end{pmatrix}$$

da cui:

$$y = bu \;\Rightarrow\; u = \frac{y}{b}.$$

Allora, sostituendo $y$ nell'espressione sopra, si ottiene

$$x = \frac{c-a}{b^2}y^2 + \frac{a}{b}y$$

che coincide esattamente con la (2.1). □

Per quanto osservato precedentemente, possiamo giungere al seguente risultato, che ci definisce precisamente la posizione di un punto critico rispetto alla funzione altezza

**Teorema 2.3** Data la successione di punti $\{Q_j\}_{j=o,\dots,3n-4} \in \mathcal{P}$, siano $Q_{k-2} = (x_{k-2}, y_{k-2})$, $Q_{k-1} = (x_{k-1}, y_{k-1}), Q_k = (x_k, y_k)$ come in 2.iii. Allora il punto critico da essi generato sulla curva spline ha le seguenti coordinate:

$$\left( \frac{(x_{k-2} - x_{k-1})(y_k - y_{k-1})^2 + (x_k - x_{k-1})(y_{k-2} - y_{k-1})^2}{(y_k - 2y_{k-1} + y_{k-2})^2} + x_{k-1}, \right.$$
$$\left. \frac{(y_k - y_{k-1})(y_{k-2} - y_{k-1})}{y_k - 2y_{k-1} + y_{k-2}} + y_{k-1} \right)$$

*Dimostrazione.* Abbiamo visto che il punto critico, in un caso del genere, si trova simultaneamente su due parabole di punti critici aventi concavità rivolte nel versi opposti. Quindi, per l'unicità, basta verificare che ne è un punto d'intersezione.
Abbiamo appena visto che le due parabole in questione sono curve di Bézier aventi come poligonali di controllo, rispettivamente:

$$Q_{k-1}, \frac{\overline{Q'Q_{k-1}}}{2}, Q_k, \quad \text{con} \quad Q' = \left( \frac{x_{k-2} - x_{k-1}}{y_{k-2} - y_{k-1}}(y_k - y_{k-1}) + x_{k-1}, y_k \right)$$

$$Q_{k-1}, \frac{\overline{Q''Q_{k-1}}}{2}, Q_{k-2}, \quad \text{con} \quad Q'' = \left( \frac{x_k - x_{k-1}}{y_k - y_{k-1}}(y_{k-2} - y_{k-1}) + x_{k-1}, y_{k-2} \right).$$

e che, per semplicità di calcolo, possiamo traslare la poligonale (tramite un'applicazione $\phi$), in modo da operare con uno dei tre punti nell'origine degli assi e considerare gli altri due arbitrari, ma tali da verificare la condizione 2.iii.
Poniano $Q_{k-2} = (\alpha_{k-2}, \beta_{k-2}), Q_{k-1} = (0,0), Q_k = (\alpha_k, \beta_k)$, dove, con abuso di notazione, $Q_{k-j} := \phi(Q_{k-j})$, e dove $\alpha_{k-j} = x_{k-j} - x_{k-1}$ e $\beta_{k-j} = y_{k-j} - y_{k-1}$, con $j = 0, 1, 2$. Consideriamo allora le due parabole di punti critici rispetto a $\varphi$ appena definite, espresse in forma parametrica :

$$\begin{pmatrix} x \\ y \end{pmatrix} = Q_{k-1}(1-u)^2 + \frac{\overline{Q'Q_{k-1}}}{2} 2u(1-u) + Q_k u^2 = \begin{pmatrix} \frac{\alpha_{k-2}\beta_k}{\beta_{k-2}} u(1-u) + \alpha_k u^2 \\ \beta_k u \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = Q_{k-1}(1-u)^2 + \frac{\overline{Q''Q_{k-1}}}{2} 2u(1-u) + Q_{k-2} u^2 = \begin{pmatrix} \frac{\alpha_k\beta_{k-2}}{\beta_k} u(1-u) + \alpha_{k-2} u^2 \\ \beta_{k-2} u \end{pmatrix}$$

e trasformiamole in forma cartesiana:

$$x = \left(\alpha_k - \frac{\alpha_{k-2}}{\beta_{k-2}}\beta_k\right)\frac{y^2}{\beta_k^2} + \frac{\alpha_{k-2}}{\beta_{k-2}}y$$

$$x = \left(\alpha_{k-2} - \frac{\alpha_k}{\beta_k}\beta_{k-2}\right)\frac{y^2}{\beta_{k-2}^2} + \frac{\alpha_k}{\beta_k}y$$

A questo punto imponiamo l'uguaglianza tra le due espressioni di $x$ per determinarne i punti d'intersezione.
Facendo semplici calcoli, otteniamo:

$$y\left(\frac{\alpha_k}{\beta_k} - \frac{\alpha_{k-2}}{\beta_{k-2}} + \frac{\alpha_{k-2}\beta_k^2 - \alpha_k\beta_k\beta_{k-2} - \alpha_k\beta_{k-2}^2 + \alpha_{k-2}\beta_{k-2}\beta_k}{\beta_{k-2}^2\beta_k^2}y\right) = 0$$

di cui una soluzione è l'origine, cioè il punto $Q_{k-1}$, che non appartiene alla curva, e l'altra, è il punto critico, di coordinate

$$\left(\frac{\alpha_{k-2}\beta_k^2 + \alpha_k\beta_{k-2}^2}{(\beta_k + \beta_{k-2})^2}, \frac{\beta_k\beta_{k-2}}{\alpha_k + \beta_{k-2}}\right).$$

Adesso basta effettuare la traslazione inversa $\phi^{-1}$, che ritrasforma il punto $(0,0)$ in $(x_{k-1}, y_{k-1})$ arbitrario, per ottenere l'espressione generica del punto critico rispetto a $\varphi$, cioè

$$\left(\frac{\alpha_{k-2}\beta_k^2 + \alpha_k\beta_{k-2}^2}{(\beta_k + \beta_{k-2})^2} + x_{k-1}, \frac{\beta_k\beta_{k-2}}{\beta_k + \beta_{k-2}} + y_{k-1}\right).$$

□

**Osservazione 2.2** Nel caso in cui i punti della poligonale, $Q_{k-2} = (x_{k-2}, y_{k-2})$, $Q_{k-1} = (x_{k-1}, y_{k-1}), Q_k = (x_k, y_k)$, siano posizionati in modo che due di loro, consecutivi, abbiano stessa altezza, ad esempio $Q_{k-2} = (x_{k-2}, y_{k-2}), Q_{k-1} = (x_{k-1}, y_k), Q_k = (x_k, y_k)$, per il teorema appena dimostrato, le coordinate del punto critico diverrebbe il seguente:

$$\begin{aligned} &\left(\frac{(x_{k-2} - x_{k-1})(y_k - y_k)^2 + (x_k - x_{k-1})(y_{k-2} - y_k)^2}{(y_k - 2y_k + y_{k-2})^2} + x_{k-1},\right. \\ &\qquad\qquad \left.\frac{(y_k - y_k)(y_{k-2} - y_k)}{y_k - 2y_k + y_{k-2}} + y_k\right) \\ = &\left(\frac{(x_k - x_{k-1})(y_{k-2} - y_k)^2}{(y_{k-2} - y_k)^2} + x_{k-1}, y_k\right) = (x_k, y_k) \end{aligned}$$

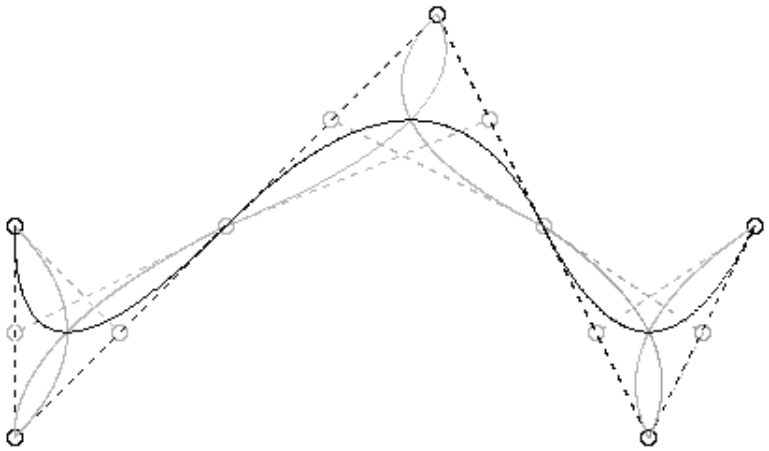


Figura 2.3: Esempio di curva spline con relativi punti critici rispetto alla funzione altezza.

e sappiamo che questo è vero quando si verificano i casi 1.i e 1.ii.
Quindi, in realtà, quanto asserito nel teorema sopra, vale per ogni punto critico della spline.

A questo punto, possiamo anche esprimere le coordinate dei punti critici esclusivamente in funzione dei punti di controllo. Basta, infatti, porre $Q_{k-2} = \frac{P_{i-1}+P_i}{2}$, $Q_{k-1} = P_i$, $Q_k = \frac{P_i+P_{i+1}}{2}$ per ottenere:

$$\left( \frac{(x_{i-1}-x_i)(y_{i+1}-y_i)^2 + (x_{i+1}-x_i)(y_{i-1}-y_i)^2}{2(y_{i+1}-2y_i+y_{i-1})^2} + x_i, \right.$$
$$\left. \frac{(y_{i+1}-y_i)(y_{i-1}-y_i)}{2(y_{i+1}-2y_i+y_{i-1})} + y_i \right),$$

che rappresentano le coordinate di ogni punto critico di una spline, con $i = 2, ..., n-2$.

## 2.2 Calcolo delle funzioni di taglia

Il calcolo della funzione di taglia di una curva richiede la localizzazione dei punti critici di quest'ultima rispetto alla funzione misurante scelta, perchè questi ne determinano tutte le caratteristiche. In particolare, nel caso della funzione altezza, i punti critici di una curva abbastanza regolare (almeno $C^1$) sono rappresentati da massimi e minimi, la cui localizzazione ci permette di dire a quali quote si crea un aumento del numero di componenti connesse per archi (minimi) a e quali una sua diminuzione (massimi).
Le ordinate di questi punti determinano le coordinate dei punti di discontinuità della funzione di taglia, cioè dei punti contenenti tutte le informazioni utili che tali funzioni possono fornirci.
Nello studio di una curva aperta, definita su un compatto e continua, si osserva che il numero di massimi e minimi o è lo stesso, o il numero dei primi è maggiore di uno rispetto ai secondi, o il numero dei secondi è maggiore di uno rispetto ai primi.
Per il calcolo della funzione di taglia si procede secondo le regole del seguente algoritmo:

1. Si ordinano i punti di minimo rispetto alla loro altezza;

2. Si prende in esame quello con ordinata più grande e lo si confronta con i massimi ad esso contigui;
   Se tale minimo è un estremo, allora esiste un solo massimo e si prende quello;
   Se ne sono due, allora:

   - se nessuno dei due è un estremo della curva, si prende quello con ordinata più piccola;
   - se uno dei due costituisce un estremo della curva, allora si prende l'altro.

3. Si eliminano le coppie (min, max) di punti scelti;

4. Si ricomincia dal punto 1.

Le ordinate delle coppie di punti $(y_{min}, y_{max})$ presi di volta in volta saranno, rispettivamente l'ascissa e l'ordinata di un punto di discontinuità della funzione di taglia.
Siano $(\alpha, \beta)$ le coordinate di un tale punto. Allora, la funzione di taglia è definita dall'unione di triangoli i cui vertici sono eprimibili dalle seguenti coordinate: $\{(\alpha, \alpha), (\alpha, \beta), (\beta, \beta)\}$ e la cui molteplicità è legata al numero di minimi che hanno come ordinata $y = \alpha$.
Ora, visto che i minimi che ci interessano effettivamente sono tutti, mentre i massimi che danno un contributo sono tutti eccetto quelli (se presenti) agli estremi, allora, il numero dei primi sarà sempre di uno più grande di quello dei secondi. Così, ad un certo punto, arriveremo al punto di minimo assoluto della curva, a cui non potremo associare nessun massimo; ebbene, in corrispondenza della sua ordinata, assoceremo una semiretta, detta retta d'angolo.
Questo algoritmo lo utilizzeremo nel programma di visualizzazione delle funzioni di taglia, che verrà trattato nel terzo capitolo.
Cf. [D'A00] per maggiori dettagli.

# Capitolo 3

# Programma di Visualizzazione

L'idea di creare un'applicazione che possa visualizzare le funzioni di taglia di curve spline è nata principalmente con lo scopo di verificare e confermare i risultati teorici ottenuti nel corso di questo lavoro di tesi. Il secondo motivo, che ci ha spinti a curarla in tutte le sue parti, è quello di poter utilizzarla effettivamente come strumento di studio. Infatti, la flessibilità strutturale del programma, permette di creare e modificare interattivamente la spline in qualsiasi modo si voglia e, contemporaneamente, le variazioni in tempo reale della funzione di taglia associata rendono possibile lo studio di tutte le configurazioni da questa assunte.
Come linguaggio per lo sviluppo dell'applicazione è stato scelto Java.
I motivi alla base di questa scelta riguardano la flessibilità e l'alto grado di evoluzione del linguaggio, che permettono un'ottima strutturazione logica dei processi, la trasparenza degli algoritmi e la semplicità nella costruzione dell'interfaccia grafica.
Tali peculiari caratteristiche hanno reso possibile una buona standardizzazione e modularizzazione dell'applicazione.

## 3.1 Funzionalità

Il programma si presenta come una finestra con due pannelli, un menù e due pulsanti.
Il pannello di sinistra è stato creato per il disegno interattivo della spline.
Questo viene generato automaticamente dalla scelta delle coordinate dei punti di controllo, che vengono visualizzati a schermo ogni qualvolta si posizioni il cursore sul pannello e si clicchi il tasto sinistro del mouse. Insieme ai punti viene creata anche la poligonale da essi generata e quindi la curva associata.
Se si desidera apportare delle modifiche alla curva, è possibile farlo in due modi diversi.
Il primo consiste nella modifica locale della spline tramite variazione delle coordinate dei punti di controllo: posizionando il cursore sul punto scelto, possiamo trascinarlo nella nuova posizione tenendo premuto il tasto sinistro del mouse.
Il secondo consiste nella possibilità di eliminare punti di controllo (cliccandovi sopra due volte) o aggiungerne ulteriori.
Alla base di questo pannello, si trovano due pulsanti: il primo permette di attivare/disattivare il disegno della poligonale di controllo; il secondo permette di visualizzare i punti critici della curva rispetto alla funzione altezza.
Il pannello di destra è stato creato per poter visualizzare la funzione di taglia della spline ed è quindi collegato al pannello di sinistra in modo tale da poter raccogliere tutte le informazioni sulle posizioni dei punti critici e sulle loro possibili modifiche. La funzione di taglia varia in tempo reale ogni qualvolta si modifica la spline.
Infine, nell'applicazione è presente un menù, in cui compaiono quattro voci: *new*, per disegnare una nuova curva; *save*, per salvare su file una curva; *load*, per caricare una curva salvata; *exit*, per chiudere l'applicazione.
Le curve spline vengono rappresentate mediante un file di tipo testuale con estensione *".spl"*. All'interno del file sono mantenute le informazioni riguardanti il grado della spline, la partizione nodale ed il numero di punti di

controllo.

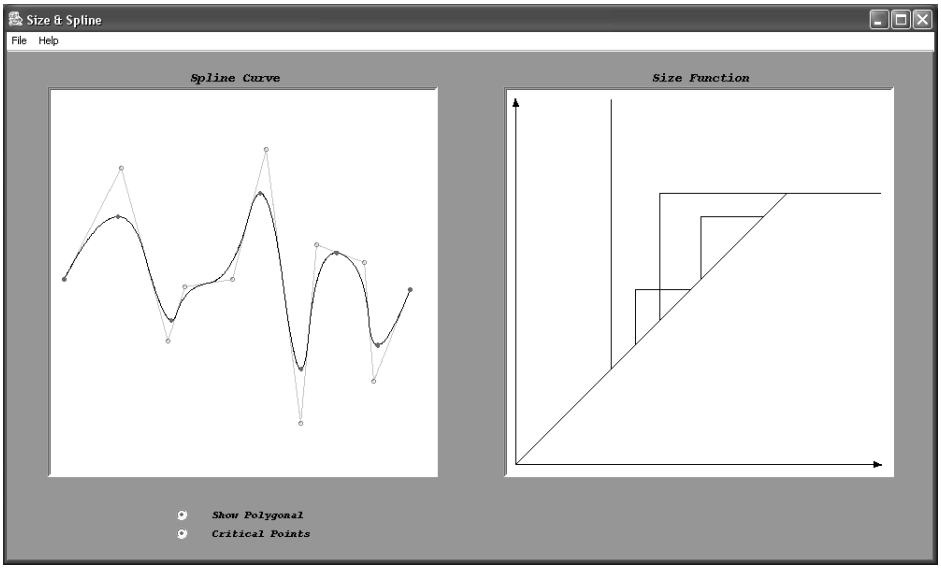



Figura 3.1: Interfaccia grafica del programma *Size & Spline*. In dettaglio: una curva spline, i relativi punti critici e la funzione di taglia associata.

## 3.2 Struttura dell'applicazione

### 3.2.1 Interfaccia grafica

L'applicazione dispone di un'interfaccia grafica composta da un frame (*Application1*), all'interno di cui sono definiti due pannelli (*MousePanel* e *SizePanel*). Il frame gestisce l'inizializzazione delle variabili principali, fra cui la curva stessa, gli oggetti per il disegno e le funzioni di caricamento/salvataggio file.

Entrambi i pannelli gestiscono un'oggetto di tipo *BufferedImage* su cui avviene il disegno vero e proprio. I processi grafici sono tuttavia delegati agli oggetti di tipo *DrawWindow*.

Il primo pannello (*MousePanel*) gestisce il disegno interattivo della curva spline corrente. In particolare si occupa di catturare gli eventi generati dal

mouse ed effettuare le chiamate opportune alle funzioni di disegno e di aggiornamento delle strutture dati.
I due pulsanti alla base dell'applicazione interagiscono con il frame causando il lancio di trap catturati dall'oggetto *MousePanel*.
Il disegno simultaneo su entrambi i pannelli (Curva Spline e Funzione di Taglia) è realizzato semplicemente passando una *handle* di *SizePanel* a *MousePanel* durante la fase di inizializzazione.
Data l'eterogeneità delle componenti, lo stile del frame è un *GridBagLayout*, realizzato mediante l'ausilio di un *Wizard* visuale.

### 3.2.2 Metodi di disegno

La gestione del disegno su entrambi i pannelli viene svolta da una classe di oggetti denominata *DrawWindow*. Tali oggetti vengono generati su una *BufferedImage* in cui viene effettivamente visualizzato il risultato delle loro elaborazioni.
La classe suddetta è stata pensata, per quanto possibile, indipendente dalle entità da sottoporre all'elaborazione grafica; in particolare, tramite la generazione di una tassonomia delle classi di oggetti 2D con unico capostipite la classe astratta *Draw2D*, è stato possibile strutturare un processo algoritmico generale per la gestione del disegno.
Come sappiamo, in generale, il disegno di un qualsiasi oggetto al computer viene effettuato per accensione di pixel; ne è, quindi, per sua natura, un'approssimazione discreta.
Ma come si fa a passare dal continuo al discreto?
Ci sono sostanzialmente due modi. Il primo è un metodo che si basa sull'indicizzazione dei pixel; l'altro, molto più efficiente, consiste nel costruire una mappa che trasforma il "sistema di coordinate del mondo" (*Window*) nel sistema di coordinate schermo (*Viewport*). Questo secondo metodo, che abbiamo usato nel programma *Size & Spline*, consiste nel definire un *mapping W/V*, cioè una mappa che manda ogni punto della *Window*, sistema

cartesiano a coordinate reali orientato nel modo convenzionale, in un punto della *Viewport*, sistema a coordinate intere ribaltato rispetto all'asse delle ascisse (cioè con l'asse delle $y$ rivolto verso il basso).
Ci si aspetta che, nel passaggio da un sistema all'altro, l'oggetto non subisca deformazioni, cioè che un punto $(x_w, y_w)$ nella *Window* mantenga le stesse proporzioni sia nelle $x$ che nelle $y$ quando viene effettuata la trasformazione in $(x_v, y_v)$ nella *Viewport*. Si impone perciò la validità delle relazioni seguenti:

$$\begin{cases} \frac{x_w - x_{w_{min}}}{x_{w_{max}} - x_{w_{min}}} & = & \frac{x_v - x_{v_{min}}}{x_{v_{max}} - x_{v_{min}}} \\ \frac{y_{w_{max}} - y_w}{y_{w_{max}} - y_{w_{min}}} & = & \frac{y_{v_{max}} - y_v}{y_{v_{max}} - y_{v_{min}}} \end{cases}$$

dove $y_{w_{max}} - y_w = y_w - y_{w_{min}}$, $y_{v_{max}} - y_v = y_v - y_{v_{min}}$.
Da tali relazioni, si ricavono le coordinate del punto $(x_v, y_v)$, cioè:

$$\begin{cases} x_v & := & round(S_x(x_w - x_{w_{min}}) + x_{w_{min}}) \\ y_v & := & round(S_y(y_{w_{max}} - y_w) + y_{v_{max}}) \end{cases}$$

dove:
$S_x = \frac{x_{v_{max}} - x_{v_{min}}}{x_{w_{max}} - x_{w_{min}}}$ è il rapporto di scala fatto sull'asse $x$,
$S_y = \frac{y_{v_{max}} - y_{v_{min}}}{y_{w_{max}} - y_{w_{min}}}$, è il rapporto di scala fatto sull'asse $y$,
e *round* rappresenta l'approssimazione agli interi.
Per evitare che l'oggetto si deformi in seguito al *mapping W/V*, basta richiedere una trasformazione di scala uniforme, cioè $S_x = S_y$.
Nel nostro problema, l'oggetto da disegnare è una curva espressa tramite equazione parametrica $C(u), u \in [0, 1]$. Bisogna discretizzarla senza renderla discontinua; bisogna, cioè, evitare che si creino dei "buchi" quando la si calcola in due istanti di tempo molto vicini a cui sono associati due valori abbastanza distanti. Inoltre, si richiede anche che non si creino addensamenti di punti. Si è, quindi, ritenuto opportuno utilizzare un algoritmo adattivo che imponga una distanza limitata tra due punti della curva relativi a due istanti di tempo successivi.
Vogliamo disegnare la spline in modo che $C(u_i)$ sia distante da $C(u_{i-1})$ di una quantità $D$ tale che:

$$EX2 \leq D \leq EX1,$$

dove fissiamo $EX2 = 1$ ed $EX1 = \sqrt{2}$.
Usiamo, quindi, il seguente algoritmo:

```
valuta C(u[i - 1]);
```

Si calcola $C(u_{i-1})$

```
n := 0;
```

Si inizializza n a 0

```
u[i] := u[i - 1] + du;
```

Si calcola l'incremento tra due istanti successivi

```
repeat
   n := n + 1;
```

Si incrementa n

```
   valuta C(u[i]);
```

Si calcola $C(u_i)$

```
   D := ||C(u[i]) - C(u[i - 1])||_2;
```

Si calcola la distanza in norma 2

```
   k := 0;
```

Si fissa k = 0 e si analizzano i due casi in cui D non è compreso tra i valori voluti

```
  if D > EX1 then
  begin
     k := -1;
     u[i] := u[i] + k*du * 2^(-n);
```

Se $D > \sqrt{2}$, avviciniamo $u_i$ ad $u_{i-1}$

```
  end;
  if D < EX2 then
  begin
     k := 1;
     u[i] := u[i] + k*du * 2^(-n);
```

Se $D < 1$, allontaniamo $u_i$ da $u_{i-1}$

```
  end;
until (k = 0);
```

Si prosegue finchè k diventa pari a 0
`du := u[i] - u[i - 1];`
Si prende il nuovo incremento
`i := i + 1;`
Si ricomincia dall'inizio.
Tale algoritmo, "adattando" di volta in volta la parametrizzazione del dominio alla struttura della curva, permette una scelta appropriata dei punti da mappare.
Nell'applicazione pratica è tuttavia necessario imporre dei vincoli al processo di riduzione dei sottointervalli, in modo da evitare la divergenza della computazione nel caso di curve con ampie variazioni.

### 3.2.3 Oggetti geometrici

L'applicazione tratta tre tipi di oggetti geometrici: i segmenti, le circonferenze e le curve spline. Per ognuno di questi esiste una classe opportuna (*Segment2D*, *Circle* e *Curve2D*), istanza della classe astratta *Draw2D*. Il metodo derivato dalla classe "padre" è quello riguardante la valutazione parametrica della varietà in un punto ed è stato reimplementato in ognuna delle tre.
In particolare la classe *Curve2D* adotta lo schema di valutazione iterativo spiegato nel primo capitolo. Tale classe, inoltre, incapsula i dati relativi ad una curva spline mediante la creazione di un'istanza dell'oggetto *SplineCurve*, i cui metodi permettono di effettuare operazioni sull'insieme dei punti di controllo e sulla partizione nodale della spline in modo trasparente.
Gli elementi base elaborati da tutte e tre queste classi sono appunto coppie razionali (*DoublePoint*) e punti di controllo (*CP*).
Il programma dispone della possibilità di caricare/salvare curve realizzate mediante i due metodi *load* e *save* della classe *SplineCurve*.

### 3.2.4 Calcolo delle funzioni di taglia

La possibilità di visualizzare la funzione di taglia di una spline ha richiesto la costruzione di un vettore di punti critici secondo le modalità presentate nel secondo capitolo. Questo vettore presenta caratteristiche un pò diverse rispetto a quanto fatto con l'applicazione dell'algoritmo di knot-refinement ma, sostanzialmente, i risultati prodotti sono gli stessi (vedi appendice A).
Nella classe *DrawWindow* questo vettore è stato poi sottoposto ad alcune operazioni che permettono di accendere i pixel dei punti critici e spegnerli quando, in seguito a modifiche della curva, non facciano più parte del vettore.
Per quanto riguarda il calcolo effettivo della funzione di taglia, è stato utilizzato l'algoritmo presentato alla fine del capitolo precedente.

### 3.2.5 Schema dei moduli

Riportiamo di seguito lo schema delle classi del programma *Size & Spline* realizzato mediante diagrammi **UML**.
Si è scelto di presentare i diagrammi relativi alla fase di disegno e alla gerarchia delle classi negli oggetti geometrici.
L'unione dei tre schemi rappresenta il nucleo concettuale su cui è stato strutturato il programma, e mette in evidenza l'indipendenza fra le fasi di elaborazione, disegno e visualizzazione.
Complessivamente il programma è composto da tredici classi, nasce come applicazione locale e, pur avendo un necessario livello di ridondanza nelle funzionalità, presenta un alto grado di indipendenza e modularità.

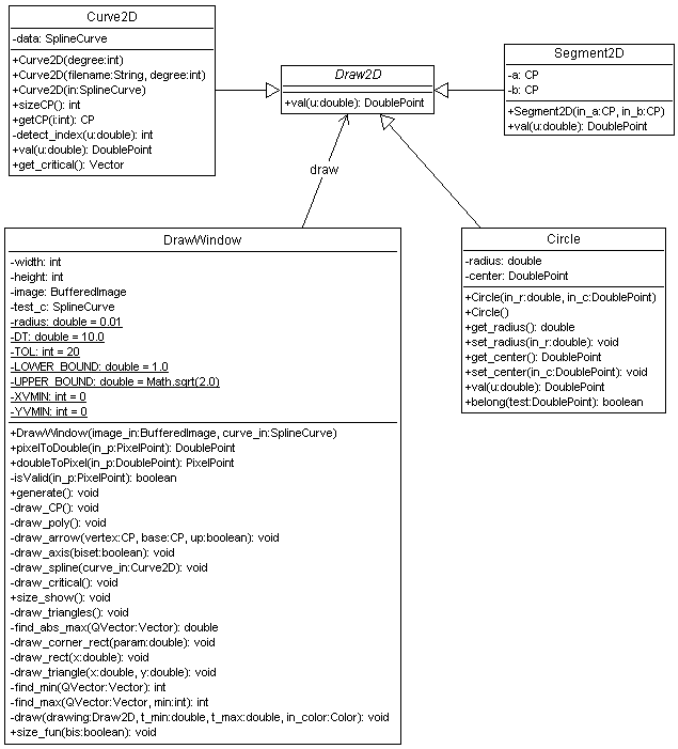



Figura 3.2: Diagramma relativo alle classi coinvolte nel disegno.

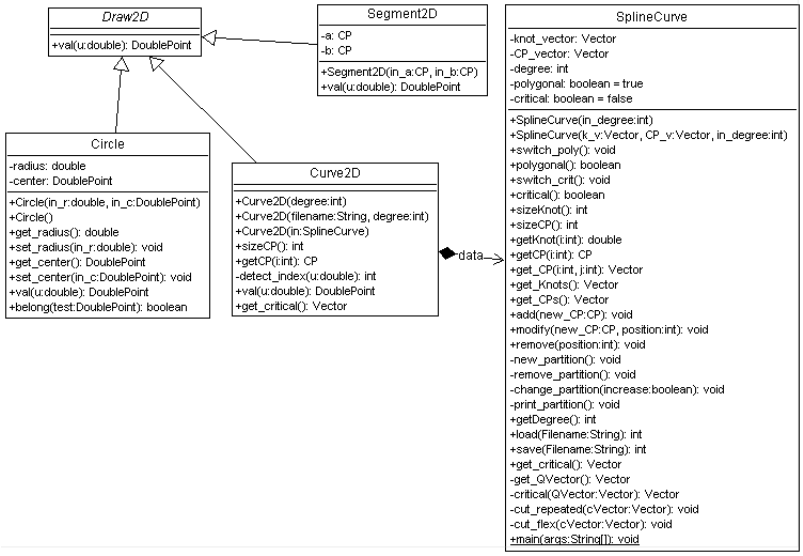



Figura 3.3: Diagramma rappresentante le relazioni di ereditarietà tra le classi relative agli oggetti geometrici.

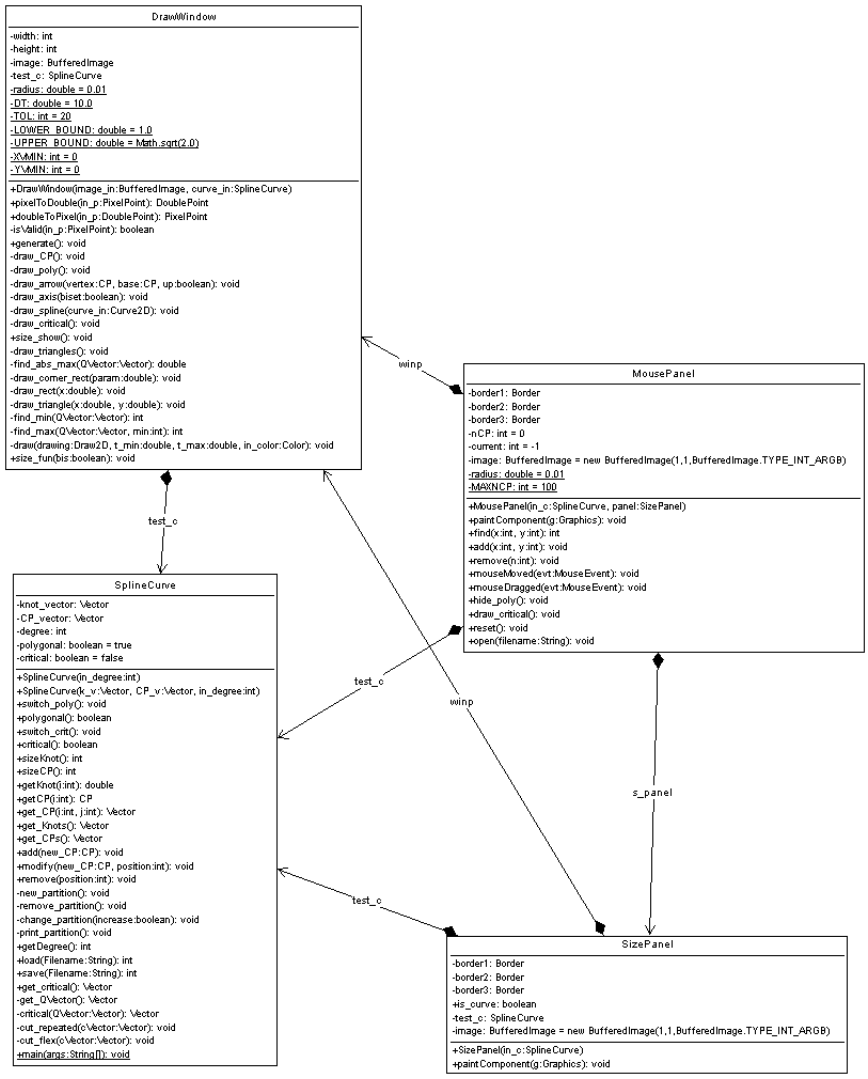



Figura 3.4: Diagramma relativo alle relazioni fra i due pannelli, la classe di disegno e i dati della curva rappresentata.

# Conclusioni

L'idea di studiare le funzioni di taglia su curve spline nasce dalla consapevolezza della rilevanza di queste ultime come primitive geometriche sia in campo di modellazione, che di ricostruzione delle immagini, con applicazioni estese in vari campi (software d'animazione, disegno assistito al calcolatore).
Il problema affrontato in questo lavoro di tesi, sia da un punto di vista teorico, che pratico, ci ha fornito come risultato principale la localizzazione delle coordinate dei punti critici, dipendenti solamente dalle caratteristiche locali della curva.
Per quanto riguarda gli sviluppi futuri, sicuramente risulta interessante estendere lo studio delle funzioni di taglia alle curve spline di qualsiasi grado, ed in seguito alle superfici.
Attualmente, è già stato osservato che i risultati raggiunti valgono anche per le curve chiuse periodiche di secondo grado, benchè non ci sia stata ancora una formulazione definitiva.
Nell'immediato, un altro importante obiettivo sarà quello di portare la rappresentazione logica dei dati su un supporto SVG.

# Appendice A

# *Size & Spline*

In questa appendice viene presentato il codice del programma sviluppato. Il codice non sarà esposto nella sua completezza, ma verranno presentate solamente le classi e i relativi metodi, insieme all'implementazione di alcune parti ritenute significative.

Classi relative ai due pannelli di visualizzazione.

Primo pannello.

```
public class MousePanel extends JPanel
  implements MouseMotionListener {
  public MousePanel(SplineCurve in_c, SizePanel panel){}
```

Metodo che gestisce il disegno

```
  public void paintComponent(Graphics g){}
```

Trova la posizione del mouse e cambia forma se ci si trova sopra

```
  public int find(int x,int y){
```

Dobbiamo fargli verificare che il mouse non sia all'interno di un CP

```
  boolean cond;
  CP temp_cp;
  DoublePoint center;
  Circle temp_circle;
  DoublePoint mouse_pos;
  PixelPoint mouse_screen = new PixelPoint(x,y);
```

```
  mouse_pos = winp.pixelToDouble(mouse_screen);
  for (int i = 0; i < test_c.sizeCP(); i++){
```

Creiamo il cerchio razionale corrispondente all'i-esimo CP

```
    temp_cp = test_c.getCP(i);
    center = new DoublePoint(temp_cp.get_x(),temp_cp.get_y());
    temp_circle = new Circle(radius,center);
```

Eseguiamo il test di appartenenza

```
    cond= temp_circle.belong(mouse_pos);

    if (cond) return i;
  }
  return -1;
  }

  public void add(int x, int y){

    if (nCP < MAXNCP){
    PixelPoint temp_pixel = new PixelPoint(x,y);
    DoublePoint temp_p = winp.pixelToDouble(temp_pixel);
    CP val = new CP(temp_p.get_x(),temp_p.get_y());
    test_c.add(val);

    current = nCP;
    nCP++;
    repaint();
    }
  }

  public void remove(int n){

    if (n < 0 || n >= nCP) return;

    test_c.remove(n);
    nCP--;
    if (current == n) current = -1;
    repaint();
    }

  public void mouseMoved(MouseEvent evt){
    int x = evt.getX();
    int y = evt.getY();

    if(find(x,y) >= 0)
 setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
```

```
    else setCursor(Cursor.getDefaultCursor());
    }

  public void mouseDragged(MouseEvent evt){
    int x = evt.getX();
    int y = evt.getY();

    if (current >= 0){
      PixelPoint temp_pixel = new PixelPoint(x,y);
      DoublePoint temp_p = winp.pixelToDouble(temp_pixel);
```

Modifico il cp nella Spline

```
      CP cp = new CP(temp_p.get_x(),temp_p.get_y());
      test_c.modify(cp,current);
      repaint();
      }
  }

  public void hide_poly(){}

  public void draw_critical(){}

  public void reset(){}

  public void open(String filename){}
```

Variabili e costanti

```
  private int nCP = 0;
  private int current = -1;
  private SizePanel s_panel;
  private SplineCurve test_c;
  private BufferedImage image =
      new BufferedImage(1,1,BufferedImage.TYPE_INT_ARGB);
  private DrawWindow winp = new DrawWindow(image,test_c);
  private static final double radius = 0.01;
  private static final int MAXNCP  = 100;
}
```

Secondo pannello.

```
public class SizePanel extends JPanel{

  public SizePanel(SplineCurve in_c){}
```

Metodo che gestisce il disegno

```
  public void paintComponent(Graphics g){}
```

Dati della classe

```
  public boolean is_curve;
  private SplineCurve test_c;
  private BufferedImage image =
      new BufferedImage(1,1,BufferedImage.TYPE_INT_ARGB);
  private DrawWindow winp = new DrawWindow(image,test_c);
  }
```

Classe in cui avviene la visualizzazione.

```
public class DrawWindow {
```

Variabili

```
       private int width, height;
       private BufferedImage image;
       private SplineCurve test_c;
```

Costruttore

```
public DrawWindow(BufferedImage image_in,
                          SplineCurve curve_in){}
```

Effettua il mapping viewport/window

```
       public DoublePoint pixelToDouble(PixelPoint in_p){}
```

Metodo che effettua il mapping window/viewport

```
       public PixelPoint doubleToPixel(DoublePoint in_p){}
```

Metodo che controlla l'appartenenza di un punto alla viewport

```
        private boolean isValid(PixelPoint in_p){
            int x,y;

            boolean out_of;
            x = in_p.get_x();
            y = in_p.get_y();

            out_of = (x < 0)||(y < 0)||(x >= width)||(y >= height);

            return(!out_of);
        }
```

Metodo che genera l'immagine nel primo pannello

```
public void generate(){
  if (test_c.polygonal()){
        draw_poly();
        draw_CP();
}
  if (test_c.sizeCP() > test_c.getDegree()) {
    Curve2D main_curve = new Curve2D(test_c);
    draw_spline(main_curve);
    if (test_c.critical()) draw_critical();
  }

}
```

Metodo che disegna i punti di controllo

```
private void draw_CP(){}
```

Metodo che disegna la poligonale

```
private void draw_poly(){
 Segment2D temp_s;
 for (int h = 0; h < (test_c.sizeCP()-1); h++){
  temp_s = new Segment2D(test_c.getCP(h),test_c.getCP(h+1));
  draw(temp_s,0.0,1.0, Color.lightGray);
    }
}
```

Metodo che disegna le frecce degli assi

```
private void draw_arrow(CP vertex, CP base, boolean up){}
```

Metodo che disegna gli assi

```
private void draw_axis(boolean biset){}
```

Metodo che disegna la spline

```
private void draw_spline(Curve2D curve_in){
  draw(curve_in,0.0,1.0, Color.black);
}
```

Metodo che disegna i punti critici

```
   private void draw_critical(){

     Circle temp_c;
     DoublePoint center;
```

```
        Vector critical = test_c.get_critical();
        for (int h = 0; h < (critical.size()); h++){
          center = (DoublePoint)critical.elementAt(h);
          double rad = 0.01;
          temp_c = new Circle(rad, center);
          draw(temp_c,0.0,1.0,Color.magenta);
          double eps = 0.001;
          for(int i = 2; i < 8; i++){
            temp_c = new Circle((2*rad)/ i, center);
            draw(temp_c,0.0,2.0,Color.magenta);
        }
        }
      }
```

Metodo che disegna la funzione di taglia

```
public void size_show(){
  size_fun(false);
  draw_triangles();
}
```

Metodo che disegna i triangoli

```
 private void draw_triangles(){
   Vector critical = test_c.get_critical();

   int size = critical.size();
   int min,max;
   double x,y;
```

Disegno la retta parallela all'asse delle x

```
   double corner;
   corner = find_abs_max(critical);
   draw_corner_rect(corner);

   while(size > 2){
    min = find_min(critical);
    max = find_max(critical,min);
    x = ((DoublePoint)critical.elementAt(min)).get_y();
    y = ((DoublePoint)critical.elementAt(max)).get_y();
    draw_triangle(x,y);
    critical.removeElementAt(min);
    if(min < max) max = min;
    critical.removeElementAt(max);
    size = critical.size();
   }
   if (size == 2) min = find_min(critical);
```

```
        else min = 0;
      x = ((DoublePoint)critical.elementAt(min)).get_y();
      draw_rect(x);
    }
```

Metodo che trova il massimo assoluto

```
double find_abs_max(Vector QVector){
  double max;
  double temp;
  max = ((DoublePoint) QVector.elementAt(0)).get_y();

  for(int i=0; i < QVector.size(); i++){
   temp =((DoublePoint) QVector.elementAt(i)).get_y();
   if (max < temp) max = temp;
  }
  return max;
}

private void draw_corner_rect(double param){

  Segment2D semi_bis, max_rect, semi_bis2;

semi_bis = new Segment2D(new CP(-0.94,-0.94),new CP(param,param));
semi_bis2 = new Segment2D(new CP(param,param), new CP(-0.94,-0.94));
max_rect = new Segment2D(new CP(param,param),new CP(0.94,param));
draw(semi_bis,0.0,1.0, Color.black);
draw(max_rect,0.0,1.0, Color.black);
draw(semi_bis2,0.0,1.0, Color.black);
}
```

Metodo che disegna la retta d'angolo

```
  private void draw_rect(double x){
    CP start = new CP(x,x);
    CP end = new CP(x,0.94);
    Segment2D angle_line = new Segment2D(start,end);
    draw(angle_line, 0.0,1.0, Color.black);
  }
```

Metodo che disegna un triangolo

```
  private void draw_triangle(double x, double y){
    CP vertex = new CP(x,y);
    CP v_1 = new CP(x,x);
    CP v_2 = new CP(y,y);

    Segment2D c_1 = new Segment2D(vertex, v_1);
```

```
    Segment2D c_1_b = new Segment2D(v_1, vertex);
    Segment2D c_2 = new Segment2D(vertex, v_2);
    Segment2D c_2_b = new Segment2D(v_2, vertex);
    draw(c_1,0.0,1.0,Color.black);
    draw(c_2,0.0,1.0,Color.black);
    draw(c_1_b,0.0,0.1,Color.black);
    draw(c_2_b,0.0,0.1,Color.black);
  }
```

Metodo che trova il massimo dei minimi nel vettore dei punti critici
n.b.: ritorna l'indice dell'elemento nel vettore

```
  private int find_min(Vector QVector){
    int result;
```

Individuo le posizioni dei minimi nel vettore

```
    double first, second;

    first  = ((DoublePoint)QVector.elementAt(0)).get_y();
    second = ((DoublePoint)QVector.elementAt(1)).get_y();

    int start, end;
    if (first < second){
      start = 0;
      if ((QVector.size()%2) == 0)
        end = QVector.size()-2;
      else
        end = QVector.size()-1;
    }
    else{
      start = 1;
      if ((QVector.size()%2) == 0)
        end = QVector.size()-1;
      else
        end = QVector.size()-2;
    }
```

Fra i minimi individua la posizione del massimo

```
    double min = ((DoublePoint)QVector.elementAt(start)).get_y();
    double test;
    result = start;
    for (int i = start; i <= end; i+=2){
      test =((DoublePoint)QVector.elementAt(i)).get_y();
       if (test > min) {
         min = test;
         result = i;
```

```
      }
    }

    return result;
  }
```

Metodo che trova il minimo dei massimi adiacenti il punto passato in input
n.b.: ritorna l'indice dell'elemento nel vettore

```
  private int find_max(Vector QVector, int min){
   int result = -1;
   int pred,succ;
   pred = succ = -1;

   if (min != 0) pred = min-1;
     else result = min+1;
   if (min != (QVector.size()-1)) succ = min + 1;
     else result = min-1;

   if ((pred != -1)&&(succ!=-1)){
     double a,b;
     a = ((DoublePoint)QVector.elementAt(pred)).get_y();
     b = ((DoublePoint)QVector.elementAt(succ)).get_y();
     if (a < b) result = pred;
       else result = succ;
   }
   return result;
  }
```

Metodo che esegue effettivamente l'accensione dei pixel

```
  private void draw(Draw2D drawing, double t_min,double t_max,
                    Color in_color){

  WritableRaster raster = image.getRaster();
  ColorModel model      = image.getColorModel();

  Color f_color = in_color;
  int argb = f_color.getRGB();
  Object Color_Data = model.getDataElements(argb,null);

  PixelPoint temp_p;
```

Calcolo i Pixel da accendere

```
  DoublePoint p_pred = new DoublePoint();
  DoublePoint p_succ = new DoublePoint();
```

```
PixelPoint pix_pred;
PixelPoint pix_succ;

double t_pred = t_min;
double t_succ;
double dt = (t_max-t_min)/DT;

double k;
int n;

double distance;
```

Valuto la curva nel primo punto

```
p_pred = drawing.val(t_pred);

while(t_pred < t_max){
  pix_pred = doubleToPixel(p_pred);
  n=0;
  t_succ = t_pred + dt;
  do{
    n++;
    p_succ = drawing.val(t_succ);
```

Valuto il pixel corrispondente

```
    pix_succ = doubleToPixel(p_succ);
```

Calcolo la distanza

```
    distance = pix_pred.distance(pix_succ);

    k = 0.0;
```

Se la distanza è maggiore di quella concessa mi riavvicino a $t_{pred}$

```
    if ( distance > UPPER_BOUND){
      k = -1.0;
      t_succ = t_succ + k*dt*Math.pow(2.0,(-1*n));
      }
```

Se la distanza è minore di quella concessa mi allontano da $t_{pred}$

```
    if ( distance < LOWER_BOUND){
      k = 1.0;
      t_succ = t_succ + k*dt*Math.pow(2.0,(-1*n));
      }
  }while((k != 0.0)&&(n!=TOL));
```

```
        dt = t_succ - t_pred;
```

Disegna il punto

```
        pix_pred = doubleToPixel(p_pred);
        raster.setDataElements(pix_pred.get_x(),
                               pix_pred.get_y(),Color_Data);
```

Eseguo gli aggiornamenti

```
        t_pred = t_succ;
        p_pred.update(p_succ);
        if ((t_pred + dt) > t_max) t_pred = t_max;
        }
      }

      public void size_fun(boolean bis){
        draw_axis(bis);
      }

      private static final double radius = 0.01;

      private static double DT = 10.0;
      private static int TOL = 20;
```

In modo da avere una precisione al pixel.

```
      private static final double LOWER_BOUND = 1.0;
      private static final double UPPER_BOUND = Math.sqrt(2.0);

      private static final int XVMIN = 0;
      private static final int YVMIN = 0;

}
```

Classi relative ai pixel, ai punti di controllo e ai vettori razionali.

Pixel.

```
public class PixelPoint {
        private int x;
        private int y;

        public PixelPoint(){}

        public PixelPoint(int in_x, int in_y){}
```

```
        public void set_x(int in_x){}

        public int get_x(){}

        public void set_y(int in_y){}

        public int get_y(){}

        public double distance(PixelPoint p){
                double x_p = (double)p.get_x();
                double y_p = (double)p.get_y();

                double f1 = Math.pow((get_x() - x_p),2);
                double f2 = Math.pow((get_y() - y_p),2);

                return (Math.sqrt(f1+f2));
        }
```

Questo metodo trova il punto del vettore più distante da questo PixelPoint

```
        public PixelPoint most_far(Vector v){
                int max_distance = 0;
                int new_distance;
                int dx,dy;
                PixelPoint result = new PixelPoint();
                for (int i = 0; i < v.size();i++){
                        PixelPoint temp =(PixelPoint)v.elementAt(i);
                        dx = get_x() - temp.get_x();
                        dy = get_y() - temp.get_y();
                        new_distance = dx*dx + dy*dy;
                        if (new_distance > max_distance){
                           max_distance = new_distance;
                           result.set_x(temp.get_x());
                           result.set_y(temp.get_y());
                        }
                }
                return result;
        }

}
```

Punti di controllo.

```
public class CP {
        private double x;
        private double y;

        public CP(){}
```

```
        public CP(double in_x, double in_y){}

        public void set_x(double in_x){}

        public double get_x(){}

        public void set_y(double in_y){}

        public double get_y(){}

}
```

Punti razionali.

```
public class DoublePoint {
        private double x;
        private double y;
```

Costruttore Vuoto

```
        public DoublePoint(){}
```

Costruttore con Valori

```
        public DoublePoint(double in_x, double in_y){}

        public void set_x(double in_x){}

        public double get_x(){}

        public void set_y(double in_y){}

        public double get_y(){}

        public void update(DoublePoint p){
                set_x(p.get_x());
                set_y(p.get_y());
        }

        public double distance(DoublePoint p){
                double x_p = p.get_x();
                double y_p = p.get_y();

                double f1 = Math.pow((get_x() - x_p),2);
                double f2 = Math.pow((get_y() - y_p),2);

                return (Math.sqrt(f1+f2));
```

```
        }
}
```

Classi relative agli oggeti geometrici.

Classe astratta capostipite.

```
public abstract class Draw2D{
```

Metodo astratto

```
 public DoublePoint val(double u){
    return(new DoublePoint());
  }
}
```

Segmento

```
public class Segment2D extends Draw2D{
        private CP a;
        private CP b;

        public Segment2D(CP in_a, CP in_b){}

        public DoublePoint val(double u){
                double x,y;
                x = (1-u)*a.get_x() + u*b.get_x();
                y = (1-u)*a.get_y() + u*b.get_y();

                DoublePoint result = new DoublePoint(x,y);
                return result;
                }
}
```

Circonferenza

```
public class Circle extends Draw2D{

        private double radius ;
        private DoublePoint center ;

        public Circle (double in_r, DoublePoint in_c){}

        public Circle (){}

        public double get_radius(){}

        public void set_radius(double in_r){}
```

```
        public DoublePoint get_center(){}

        public void set_center(DoublePoint in_c){
                center = new DoublePoint(in_c.get_x(),in_c.get_y());
        }

        public DoublePoint val (double u){

                DoublePoint result = new DoublePoint();

                double x,y;
                double theta = 2.0*u*Math.PI;
                x = get_radius()*Math.cos(theta) + center.get_x();
                y = get_radius()*Math.sin(theta) + center.get_y();
                result.set_x(x);
                result.set_y(y);
                return result;
        }

        public boolean belong (DoublePoint test){

                boolean result;

                double fat1 = Math.pow (
                        (test.get_x() - center.get_x()),2);
                double fat2 = Math.pow (
                        (test.get_y() - center.get_y()),2);
                double fat3 = Math.pow (get_radius(),2);
                result = (fat1 + fat2 <= fat3);
                return result;
        }
}
```

Curva Spline.

```
public class Curve2D extends Draw2D{

                private SplineCurve data;

        public Curve2D(int degree){}

        public Curve2D(String filename,int degree){}

        public Curve2D(SplineCurve in){}

        public int sizeCP(){}
```

```
        public CP getCP(int i){}
```

Metodo per l'individuazione dell'indice dell'ultimo punto a sinistra di t nella partizione nodale

```
        private int detect_index(double u){

                int start = 0;
                int end = data.sizeKnot()-1;
                int half ;

                while (start < (end-1)){
                        half = (end+start+1)/2;

                        if ( u <= data.getKnot(half)) end = half;
                        else start = half;
                }
                if (u == 0.0) return data.getDegree();
                else return start;
        }
```

Metodo per la valutazione di una spline in un punto t sfruttando la definizione ricorrente

```
        public DoublePoint val(double u){
```

j è l'indice piu' a sinistra di t

```
                int j = detect_index(u);

                Vector read  ;
                Vector write ;

                CP P_1;
                CP P_2;
                double u_i_m, u_i;
                double x,y;
                double modulus;
                double fact1,fact2;
                CP insert_P;
```

offset mi serve per poter conservare l'iterazione $j - m - -- > j$

```
                int offset = j-data.getDegree();

                read = data.get_CP(offset,j);

                for (int m = data.getDegree(); m > 0; m--){
                                write = new Vector();
                        for (int i = j-m; i < j; i++){
```

```
                                P_1 = (CP) read.elementAt(i-offset);
                                P_2 = (CP) read.elementAt(i+1-offset);
                                u_i_m = data.getKnot(i+m+1);
                                u_i = data.getKnot(i+1);

                                double den = (u_i_m - u_i);
                                if (den == 0.0) modulus = 0.0;
                                  else modulus = 1.0/den;
                                  fact1 = u_i_m - u;
                                  fact2 = u - u_i ;

                           x = modulus*(fact1*P_1.get_x() + fact2*P_2.get_x());
                           y = modulus*(fact1*P_1.get_y() + fact2*P_2.get_y());

                                     insert_P = new CP(x,y);
                                     write.addElement(insert_P);

                                }
                        read = write;
                        offset++;
                        }
                CP result_P =   (CP) read.elementAt(0);
                DoublePoint result =
                      new DoublePoint(result_P.get_x(),result_P.get_y());
                return result;
                }

        public Vector get_critical(){}
}
```

Classe relativa ai dati di una curva spline

```
public class SplineCurve{
```

Dati

```
        private Vector knot_vector;
        private Vector CP_vector;
        private int degree;
        private boolean polygonal = true;
        private boolean critical = false;
```

Costruttore Vuoto $->$ crea una spline senza CP e Nodi

```
        public SplineCurve(int in_degree){}
```

Costruttore $->$ crea una spline prendendo nodi e CP da due vettori

```
        public SplineCurve(Vector k_v, Vector CP_v, int in_degree){}
```

Questo metodo attiva/disattiva il disegno della poligonale

```
public void switch_poly(){}
```

Questo metodo dice se la poligonale deve essere disegnata o no

```
public boolean polygonal(){}
```

Questo metodo disegna o cancella i punti critici

```
public void switch_crit(){}
```

Questo metodo dice se i punti critici devono essere disegnati o no

```
public boolean critical(){}

public int sizeKnot(){}

public int sizeCP(){}

public double getKnot(int i){}

public CP getCP(int i){}
```

Ritorna un vettore contente i CP dal i-simo al j-simo

```
public Vector get_CP(int i, int j){}

public Vector get_Knots(){}

public Vector get_CPs(){}
```

Aggiunge alla spline un nuovo CP

```
public void add(CP new_CP){}
```

Modifica la spline in base al nuovo CP

```
public void modify(CP new_CP, int position){}
```

Elimina dalla spline un CP

```
public void remove(int position){}

private void new_partition(){}

private void remove_partition(){}
```

```
      private void change_partition(boolean increase){
```

Modifica la partizione nodale

```
        if (increase)
                  knot_vector.addElement(new Double(1.0));
        else      knot_vector.removeElementAt(0);

                  for(int i = 0; i < getDegree(); i++){
                    knot_vector.setElementAt(new Double(0.0),i);
                  }

                  int size = CP_vector.size();
                  int k = (size - getDegree()) + 1;
                  double dt = 1.0/((double)(k-1));

                  double val = 0.0;
                  for (int i = 0; i < k; i++ ){
                         val =  ((double)i)*dt;
                        knot_vector.setElementAt(
                            new Double(val),(i+getDegree()));
                 }

                 for(int i = (k + getDegree());
                            i < (k + 2*getDegree()); i++)
                     knot_vector.setElementAt(new Double(1.0),i);

      }

      private void print_partition(){}
```

Ritorna il grado della spline

```
        public int getDegree(){}
```

Questo metodo carica la spline da file

```
        public int load(String Filename){}
```

Questo metodo salva su file la funzione spline

```
        public int save(String Filename){}
```

Metodo che restituisce il vettore dei punti critici public Vector

```
get_critical(){}
```

Metodo che calcola il vettore dei $Q_i$

```
private Vector get_QVector(){
 Vector QVector = new Vector();

 CP p_0 = (CP)CP_vector.elementAt(0);
 QVector.addElement(p_0);
 CP p_1,p_2;
 double x_1, x_2, y_1, y_2;

 for(int i =1; i < (sizeCP()-2); i ++){
   p_1 = new CP(((CP)getCP(i)).get_x(),((CP)getCP(i)).get_y());

   x_1 = ((CP)getCP(i)).get_x();
   y_1 = ((CP)getCP(i)).get_y();
   x_2 = ((CP)getCP(i+1)).get_x();
   y_2 = ((CP)getCP(i+1)).get_y();

   p_2 = new CP(((x_1 + x_2)/2), ((y_1 + y_2)/2));
   QVector.addElement(p_1);
   QVector.addElement(p_2);
 }

 int size = sizeCP();
```

Prendo il penultimo e l'ultimo punto

```
 CP p_p = (CP)getCP(size-2);
 CP p_u = (CP)getCP(size-1);
```

Li aggiungo nel vettore, perchè non li avevo messi

```
 QVector.addElement(p_p);
 QVector.addElement(p_u);

 return QVector;
 }
```

Metodo che calcola il vettore dei punti critici

```
 private Vector critical(Vector QVector){
   Vector result = new Vector();

   double x_a,x_b,x_c,y_a,y_b,y_c;
   double x,y;
   boolean cond1,cond2, cond;

   DoublePoint start = new DoublePoint(
```

```
            ((CP)CP_vector.elementAt(0)).get_x(),
            ((CP)CP_vector.elementAt(0)).get_y());
int s = CP_vector.size()-1;

DoublePoint end = new DoublePoint(
            ((CP)CP_vector.elementAt(s)).get_x(),
            ((CP)CP_vector.elementAt(s)).get_y());
result.addElement(start);
for (int i = 1; i < (sizeCP()-1); i++){
  x_a = ((CP)QVector.elementAt(2*i - 2)).get_x();
  y_a = ((CP)QVector.elementAt(2*i - 2)).get_y();
  x_b = ((CP)QVector.elementAt(2*i - 1)).get_x();
  y_b = ((CP)QVector.elementAt(2*i - 1)).get_y();
  x_c = ((CP)QVector.elementAt(2*i)).get_x();
  y_c = ((CP)QVector.elementAt(2*i)).get_y();
```

Valuto se ci sia o meno un punto critico da calcolare

cond1: La funzione è monotona crescente

```
  cond1 = (y_a < y_b)&&(y_b < y_c);
```

cond2: La funzione è monotona decrescente

```
  cond2 = (y_a > y_b)&&(y_b > y_c);

  cond = cond1||cond2;
```

Se la funzione non è monotona calcolo il punto critico

```
  if (!cond){

    x = (x_a - x_b)*(Math.pow((y_c - y_b),2)) +
      (x_c - x_b)*(Math.pow((y_a - y_b),2));
    x /= Math.pow((y_a - (2*y_b) + y_c),2);
    x += x_b;

    y = (y_a - y_b)*(y_c-y_b);
    y /= (y_a - (2*y_b) + y_c);
    y += y_b;

    DoublePoint krit = new DoublePoint(x,y);
    result.addElement(krit);
  }

}
result.addElement(end);
cut_repeated(result);
```

```
    cut_flex(result);
    return result;
  }
```

Questo metodo elimina dal vettore i punti critici ripetuti

```
  private void cut_repeated(Vector cVector){
    int i = 0;
    boolean cond;
    DoublePoint a,b;
    double x1, x2;
    int j;
```

Valuto gli elementi a coppie

```
    while (i < (cVector.size()-1)){
      a = (DoublePoint)cVector.elementAt(i);
      b = (DoublePoint)cVector.elementAt(i+1);

      cond = (a.get_y() == b.get_y());
      j = (i+1);
      x1 = a.get_x();
      while (cond){
        x2 = b.get_x();
        cVector.removeElementAt(j);
        a.set_x((x1 + x2)/2.0);

        if (j < cVector.size()){
          b = (DoublePoint)cVector.elementAt(j);
          cond = (a.get_y() == b.get_y());
        }
        else cond = false;
      }
      i++;
    }
  }
```

Questo metodo elimina i punti di flesso dal vettore

```
  private void cut_flex(Vector cVector){
    int i = 0;
    DoublePoint a,b,c;
    boolean cond1,cond2,cond;

    if (cVector.size() > 2){
```

Valuto gli elementi a triplette

```
      while (i < (cVector.size()-2)){

        a = (DoublePoint)cVector.elementAt(i);
        b = (DoublePoint)cVector.elementAt(i+1);
        c = (DoublePoint)cVector.elementAt(i+2);

        cond1 = (a.get_y() < b.get_y())&&((b.get_y() < c.get_y()));
        cond2 = (a.get_y() > b.get_y())&&((b.get_y() > c.get_y()));

        cond = cond1||cond2;
        if(cond) cVector.removeElementAt(i+1);
        i++;
      }
    }
  }
}
```

# Bibliografia


[D'A00] M. D'Amico. A new optimal algorithm for computing size function of shapes. *CVPRIP Algorithms III, Proceedings International Conference on Computer Vision*, pages 107–110, 2000. Pattern Recognition and Image Processing, Atlantic City.

[eCL99] P. Frosini e C. Landi. Size theory as a topological tool for computer vision. *Pattern Recognition and Image Analysis*, 9(4):596–603, 1999.

[eGA98] G. Casciola e G. Amati. Metodi numerici per la grafica. Dispense del corso A.A.1997-98, giugno 1998.

[HC99] Cay S. Horstmann and Gary Cornell. *Core Java*, volume I-Fundamentals. 1999.

[HC00] Cay S. Horstmann and Gary Cornell. *Core Java*, volume II-Advanced Features. 2000.

[PT97] Les Piegl and Wayne Tiller. *The Nurbs Book*. Second edition, 1997.