Modellazione grafica computazionale

Dalla curve di Bézier alle NURBS

Anna Lamboglia M63/001219 Agostino Vitaglione M63/001214
Mario Vitaglione M63/001213

12 febbraio 2024

Indice

1 Introduzione

Negli ultimi anni, con l'avvento di Internet e delle nuove tecnologie, risulta più semplice per l'uomo avere la possibilità di raccogliere dati, diventati una risorsa preziosa per via di numerosi contesti di utilizzo. Tuttavia, è necessario porre l'accento sulla seguente problematica: "*Come è possibile disegnare un modello che descriva al meglio l'andamento dei dati?*".
Una branca del calcolo numerico consente di rispondere a tale quesito trovando metodi che permettano di scegliere il modello più opportuno a seconda delle esigenze. Campi come la *Data Science* e il *Machine learning* si basano infatti sulla modellazione e visualizzazione dei dati in 2D e 3D, fornita dallo studio del calcolo numerico.
Una rappresentazione opportuna dei dati infatti consente di comprendere sia le caratteristiche del problema sia la quantità e la qualità dei dati su cui si sta lavorando. Lo studio e l'analisi delle caratteristiche del problema consentono di effettuare considerazioni e previsioni sull'andamento dei dati. Inoltre, è opportuno valutare la qualità dei dati poiché, se si riscontrano andamenti particolari, è possibile valutare se vi sono stati errori di misurazione.

In questo elaborato si illustrano dei modelli, utilizzati in computer grafica, che permettono di dare una soluzione a questa problematica.

2 La rappresentazione dei dati

Nell'ambito numerico rappresentare i dati significa costruire un modello che li descriva in modo attendibile. È possibile distinguere due tipologie di modelli:

- *Modello interpolante*: si costruisce una curva che passa per i punti assegnati assumendo che l'errore dei dati sia trascurabile, poiché di fatto si impone il passaggio della curva per i punti indicati. Di conseguenza, è possibile trascurare l'errore sui dati.
- *Modello approssimante*: si costruisce una curva che non passa per i punti assegnati. In questo caso si assume non trascurabile l'errore sui dati.

2.1 Problema dell'interpolazione

Dati $n+1$ punti distinti $(x_i, y_i)_{i=0,\dots,n}$, si vuole costruire una funzione $f(x)$ tale che nei nodi $\{x_i\}_{i=0,\dots,n}$, siano soddisfatte certe condizioni chiamate **condizioni di interpolazione**.
Questo è un problema che occorre spesso quando si vuole tracciare il grafico di una funzione conoscendone solo il valore in un numero finito di punti.

2.1.1 Interpolazione lineare

Si considerino le funzioni $\{\varphi_i\}_{i=0,\dots,n} \in \mathbb{R}$ linearmente indipendenti definite in $[a, b]$. Siano assegnati i valori (x_i, y_i) con $x_i \in [a, b]$ tale che $x_i \neq x_j \quad \forall i \neq j$. Risolvere un problema di interpolazione lineare così posto vuol dire calcolare i coefficienti $a_0, \dots, a_n \in \mathbb{R}$ tali che la funzione

$$f(x) = \sum_{i=0}^{n} a_i \varphi_i(x) \tag{2.1}$$

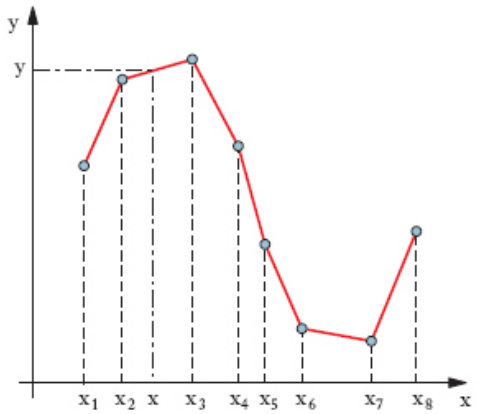


Figura 1: Interpolazione lineare a tratti con polinomi di grado 1

soddisfi le seguenti condizioni:

$$f(x_i) = y_i, \qquad i = 0, \ldots, n. \tag{2.2}$$

Tali condizioni prendono il nome di **condizioni di interpolazione**, presenti anche nelle successive interpolazioni trattate in questo paragrafo.
Si possono avere differenti tipi di interpolazione a seconda di come le funzioni φ_i sono definite. Infatti, definendo $\varphi_i(x) = x^i$, l'equazione (2.1) diventa:

$$f(x) = a_0 + a_1 x + \ldots + a_n x^n. \tag{2.3}$$

Andando quindi a imporre le condizioni di interpolazione (2.2) alla forma polinomiale (2.3), il problema della determinazione dei coefficienti del polinomio interpolante si riduce nella risoluzione del seguente sistema lineare:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{2.4}$$

dove la matrice del sistema prende il nome di **matrice di Vandermonde**.

2.1.2 Interpolazione di Lagrange

Data la funzione $f(x)$ e n+1 punti $\{x_i\}_{i=0,\ldots,n}$ per cui sono noti i valori $\{f(x_i)\}_{i=0,\ldots,n}$, il polinomio di Lagrange è così definito:

$$P(x) = \sum_{i=0}^{n} f(x_i) \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \tag{2.5}$$

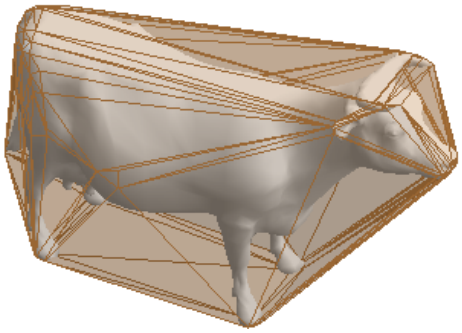


Figura 2: Esempio di inviluppo convesso in 3D

È possibile quindi scrivere il polinomio nella seguente forma:

$$P(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + ... + f(x_n)l_n(x) \tag{2.6}$$

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{2.7}$$

L'interpolazione di Lagrange è un particolare tipo di interpolazione polinomiale.

Se oltre alle condizioni di passaggio per dei punti si introducono vincoli di comportamento della funzione nell'intorno di quest'ultimi attraverso condizioni sulle derivate, si parla di **interpolazione di Hermite**; queste condizioni sono solitamente espresse in forma tabellare.

3 Punti di controllo

Bézier nel 1960 introdusse i *punti di controllo* per i problemi di interpolazione come un modo naturale per stabilire la dimensione dell'oggetto da disegnare. Fissati opportunamente i punti di controllo, si determina il più piccolo poligono convesso contenente tali punti.
Tutte le curve sono costruite a partire da questi punti di controllo e sono interamente contenute nell'inviluppo convesso. I punti di controllo sono disposti a seconda dei vincoli imposti alla curva stessa.
Per rappresentare matematicamente la curva, bisogna scegliere quali funzioni base φ_i utilizzare. Una possibile scelta è quella dei polinomi, visti nei paragrafi precedenti, poiché molto semplici da manipolare.
Ogni polinomio $p(x)$ può essere rappresentato in una di queste forme:

- **Forma Standard**: $p(x) = a_0 + a_1x + ... + a_nx^n$ con $\varphi_i(x) = x^i$.
- **Forma di Lagrange**: $p(x) = a_0l_0(x) + a_1l_1(x) + \ldots + a_nl_n(x)$ con $\varphi_i = l_i(x)$ definita in (2.7).
- **Forma di Newton**: $p(x) = a_0 + a_1(x - x_1) + \ldots + a_n(x - x_1)\ldots(x - x_n)$ con $\varphi_i(x) = \prod_{j=1}^{i}(x - x_j)$.

3.1 Considerazioni sulla forma Standard, di Lagrange e di Newton

Come accennato nel paragrafo 2.1.1, attraverso la risoluzione del sistema (2.4) è possibile determinare i coefficienti del polinomio interpolante nella forma standard.
Come dimostrato in [4], il determinante della matrice di Vandermonde V_n costruita sui nodi $x_0, ..., x_n$ è:

$$\det V_n = \prod_{0 \leq j < i \leq n} (x_i - x_j). \tag{3.1}$$

Da questo risultato è evidente che, supposti i nodi distinti, la matrice V_n è invertibile e il sistema lineare ammette una ed una sola soluzione.
Il tempo di calcolo per la risoluzione del sistema lineare con l'algoritmo di Gauss e l'algoritmo di back-substitution è $\mathcal{O}(n^3)$. Esistono ulteriori algoritmi per la risoluzione di sistemi con matrici di Vandermonde che hanno una complessità computazionale pari a $\mathcal{O}(n^2)$ e altri che calcolano la soluzione in $\mathcal{O}(n \log^2 n)$. [2]
Tuttavia, come affermato anche in [4], la matrice di Vandermonde è spesso mal condizionata. Pertanto, errori piccoli sui dati vengono amplificati drammaticamente sulla soluzione, rendendo impraticabile questa forma poiché le soluzioni calcolate sono poco attendibili.

Per quanto riguarda invece la forma di Lagrange, è possibile facilmente notare che per calcolare $l_i(x)$ (2.7) sono necessari $n-1$ prodotti; per questo motivo, il tempo di esecuzione dell'algoritmo è $\mathcal{O}(2n^2)$, risultando così computazionalmente oneroso.
Tra queste la forma che viene più usata in ambito numerico è quella di Newton, risultando più affidabile e computazionalmente migliore rispetto alla forma di Lagrange.

4 Rappresentazione di una curva

Per poter rappresentare una curva si posso utilizzare due tipologie di rappresentazione:

- *Rappresentazione non parametrica.*
- *Rappresentazione parametrica.*

4.1 Rappresentazione non parametrica

Le coordinate del punto che si muove lungo la curva sono legate attraverso una trasformazione geometrica che porta ad identificare univocamente la variabile y data la variabile indipendente x:

$$y = f(x). \tag{4.1}$$

Come affermato precedentemente, per caratterizzare una curva bisogna calcolare i coefficienti; sicché, una qualunque trasformazione geometrica applicata alla curva può essere ottenuta applicando la trasformazione al vettore dei coefficienti. Con questo tipo di rappresentazione non risulta possibile rappresentare le curve chiuse o con punti multipli.
Per esempio, è possibile rappresentare l'arco di circonferenza in figura 3 in questa forma:

$$y = \sqrt{1 - x^2} \qquad x \in [0, 1]. \tag{4.2}$$

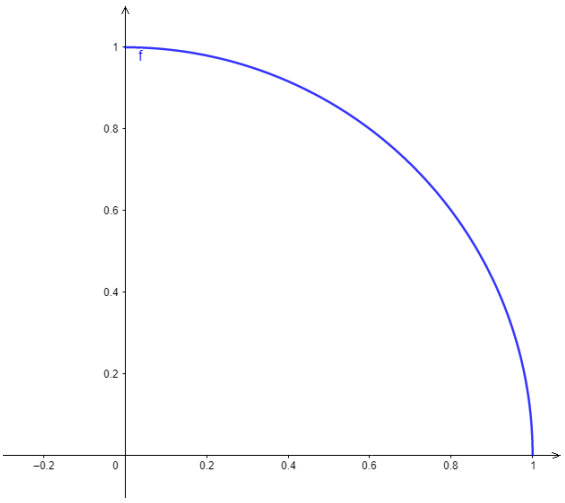


Figura 3: Rappresentazione di un arco di circonferenza in forma non parametrica

4.2 Rappresentazione parametrica

Un'equazione parametrica è un'equazione matematica in cui le variabili dipendenti e indipendenti sono a loro volta funzione di uno o più parametri:

$$C(t) = (x(t), y(t)) \qquad t \in [a, b]. \tag{4.3}$$

Questo tipo di rappresentazione è più utilizzata in computer grafica, poiché permette di disegnare curve **con punti multipli o chiuse**. Inoltre è possibile descrivere questa rappresentazione con l'utilizzo di polinomi che, come detto nel paragrafo precedente, risultano molto flessibili. Riprendendo l'esempio precedente, una circonferenza con raggio r e centro nell'origine in forma parametrica è descritta dalla seguente equazione:

$$C_r(t) = (x_r(t), y_r(t)) \qquad \begin{cases} x_r(t) = r \cdot cos(t) \\ y_r(t) = r \cdot sin(t) \end{cases} \quad t \in [0, 2\pi) \tag{4.4}$$

Si osservi che la parametrizzazione della curva non è unica. Inoltre, l'immagine di una curva viene chiamata supporto della curva. È da notare che se $t \in [0, 4\pi)$ il supporto della curva continua ad essere una circonferenza, tuttavia viene effettuato un giro in più su di essa.
Se si considera invece quest'altra rappresentazione:

$$C_r(t) = (x_r(t), y_r(t)) \qquad \begin{cases} x_r(t) = r \cdot cos(2t) \\ y_r(t) = r \cdot sin(2t) \end{cases} \quad t \in [0, \pi), \tag{4.5}$$

il supporto è sempre una circonferenza, soltanto che quest'ultima ha una velocità maggiore rispetto alla prima rappresentazione, avendo raddoppiato la frequenza delle funzioni trigonometriche.
Infine, è possibile rappresentare l'arco di circonferenza in figura 3 con l'equazione (4.4) facendo variare t in $[0, \frac{\pi}{4}]$

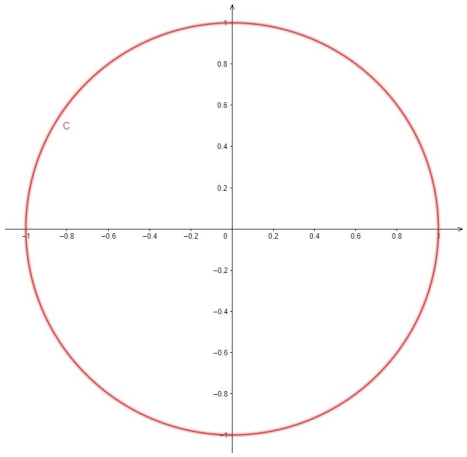


Figura 4: Rappresentazione di una circonferenza goniometrica in forma parametrica

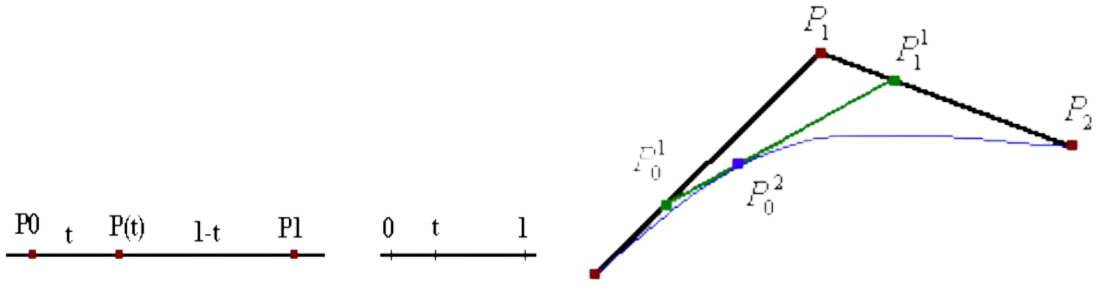
5 Curve di Bézier

Le curve di Bézier sono usate per disegnare curve nello spazio, dall'andamento morbido, a partire da un numero finito di punti. Esse sono ampiamente usate per le loro proprietà, infatti sono le prime che si possono trovare in qualsiasi programma di computer graphics poiché permettono ai designer di disegnare curve gradevoli e di qualità.

Le curve di Bézier sono delle particolari curve che devono rispettare le seguenti proprietà. Data una successione di punti $\{P_i\}_{i=0,\dots,n}$, la curva di Bézier:

- deve interpolare il primo punto P_0 e l'ultimo punto P_n;
- i suoi punti devono trovarsi interamente all'interno dell'inviluppo convesso;
- è tangente in P_0 al segmento P_0P_1;
- è tangente in P_n al segmento $P_{n-1}P_n$;
- la curvatura in P_0 dipende da $P_0P_1P_2$;
- la curvatura in P_n dipende da $P_{n-2}P_{n-1}P_n$;
- la derivata k-esima in P_0 dipende dai primi $k+1$ punti;
- la derivata k-esima in P_n dipende dagli ultimi $k+1$ punti.

Fissati $n+1$ punti di controllo, il grado del polinomio che definisce la curva di Bézier è n.
Utilizzando i polinomi visti fino a ora, si potrebbe erroneamente pensare che all'aumentare dei punti sia migliore il modello approssimante realizzato. Ciò non è così, a causa del fatto che i polinomi oscillano sempre di più all'aumentare del grado. A tal proposito, vi è un teorema di Bernstein che afferma che "dato un intervallo $[a,b]$ e fissati in esso $k+1$ punti con $k=1,2,\dots$, esiste certamente qualche funzione $f(x)$ continua su $[a,b]$ con la proprietà che la successione dei polinomi interpolanti



(a) Curva di Grado 1 (b) Curva di Grado 2

Figura 5: Curve di Bézier

$P_1(X), P_2(x), ...$, di grado pari all'indice, non converge uniformemente ad $f(x)$". [3]

Se $f(x)$ è sufficientemente regolare, è possibile definire il resto dell'interpolazione come:

$$r_n(x) = f(x) - p_n(x) \tag{5.1}$$

In [1] si dimostra il seguente teorema.

Teorema Sia $f(x) \in C^{n+1}[a,b]$ e $p(x)$ il polinomio di interpolazione di $f(x)$ relativo ai nodi $a \leq x_0 < x_1 < ... < x_n \leq b$. Per ogni $x \in [a,b]$ esiste $\xi \in (a,b)$ tale che

$$r_n(x) = \prod_{i=0}^{n}(x - x_i)\frac{f^{(n+1)}(\xi)}{(n+1)!}. \tag{5.2}$$

È possibile dimostrare che il resto cresce all'aumentare di n.
Riassumendo, le limitazioni legate all'utilizzo del polinomio interpolante sono dovute a:

- convergenza non garantita;
- legame tra grado e numero di nodi;
- aumento dell'oscillazione al crescere del grado del polinomio.

Per risolvere la problematica della convergenza del polinomio interpolante, bisogna cercare $p_n(x)$ nella classe di tutti i polinomi. È possibile utilizzare come funzioni base polinomi a tratti di grado n. Per ogni punto di controllo si costruisce un polinomio, generando in tal modo $(n+1)^2$ condizioni per costruire le curve di Bézier. Di seguito è illustrato come costruire questi particolari polinomi chiamati **polinomi di Bernstein**.
Dati due punti, P_0 e P_1, la curva di Bézier è un polinomio di grado 1 così definito:

$$B(t) = (1-t)P_0 + tP_1 \qquad t \in [0,1]. \tag{5.3}$$

Dati tre punti, P_0, P_1 e P_2, la curva di Bézier è una quadratica definita dalla seguente espressione:

$$B(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2 \qquad t \in [0,1]. \tag{5.4}$$

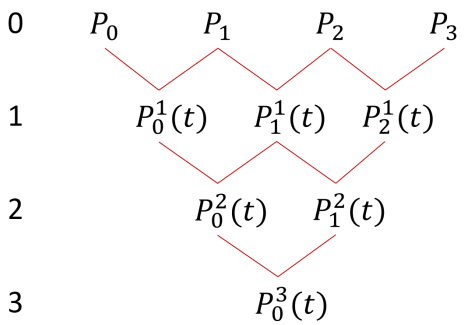


Figura 6: Iterazioni dell'algoritmo di de Casteljau per 4 punti

Generalizzando, dati un insieme di punti $\{P_i\}_{i=0,\dots,n}$ la curva di Bézier è:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} P_i (1-t)^{n-i} t^i \qquad t \in [0,1]. \tag{5.5}$$

L'ultima espressione evidenzia l'uso dei **polinomi di base di Bernstein** di grado n così definiti:

$$b_{i,n}(t) := \binom{n}{i} t^i (1-t)^{n-1} \qquad i = 0, ..., n. \tag{5.6}$$

Valutando l'equazione 5.5 agli estremi dell'intervallo $[0,1]$, si ottengono i seguenti valori:

$$B(0) = \binom{n}{0} P_0 (1-t)^n + \binom{n}{1} P_1 (1-t)^{n-1} t + ... + \binom{n}{n} P_n t^n \Big|_{t=0} = P_0 \tag{5.7}$$

$$B(1) = \binom{n}{0} P_0 (1-t)^n t + \binom{n}{1} P_1 (1-t)^{n-1} t + ... + \binom{n}{n} P_n t^n \Big|_{t=1} = P_n \tag{5.8}$$

Questi evidenziano come le curve di Bézier passano esattamente per il primo e l'ultimo punto di controllo.

5.1 Algoritmo di de Casteljau

L'algoritmo di de Casteljau è un metodo ricorsivo per valutare il polinomio della forma di Bernstein, che si dimostra essere numericamente stabile sebbene lento.
Dati 2 punti, P_0 e P_1, il segmento che li unisce è rappresentato dall'equazione parametrica (5.3).
Definiti n+1 punti $P_0, P_1, \dots, P_n$, si ottengono i polinomi intermedi:

$$P_i^j = (1-t) P_i^{j-1}(t) + t P_{i+1}^{j-1}(t) \qquad i = 0, .., n-j, \quad j = 1, .., n. \tag{5.9}$$

La curva finale di Bézier è a questo punto data da:

$$B(t) = P_0^n(t) = (1-t) P_0^{n-1}(t) + t P_1^{n-1}(t). \tag{5.10}$$

La figura 6 mostra in maniera grafica le iterazioni dell'algoritmo di de Casteljau.

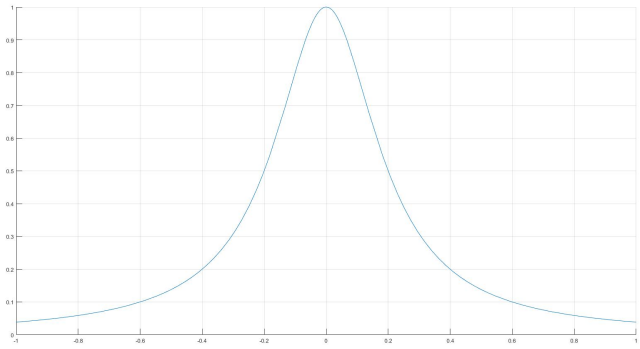


Figura 7: Grafico della funzione di Runge $f(x) = \frac{1}{1+25x^2} \quad x \in [-1, 1]$

5.2 Fenomeno di Runge

In questo paragrafo si analizza la funzione di Runge in figura 7 interpolata sia con la funzione polinomiale (2.3) sia utilizzando come funzioni base i polinomi di Bernstein.
Il fenomeno di Runge è un problema relativo all'interpolazione polinomiale in cui l'errore di interpolazione diverge al crescere del numero dei nodi:

$$\lim_{n \to +\infty} \Big(\max_{x \in [-1,1]} |f(x) - P_n(x)| \Big) = +\infty. \tag{5.11}$$

In figura 9 è possibile mettere a confronto i due tipi di interpolazione. In figura 9b, è possibile osservare come il grafico rosso approssimi bene la funzione di Runge in un intorno di 0 per poi iniziare ad oscillare fortemente agli estremi; questo situazione patologica dell'interpolazione polinomiale prendere per l'appunto il nome di **fenomeno di Runge**. Tanto più aumenta il grado del polinomio interpolante tanto più la funzione oscilla, facendo tendere l'errore d'interpolazione ad infinito.
Nella funzione in verde questo problema non si riscontra, confermando l'analisi precedente in merito all'interpolazione coi polinomi di Bernstein. In questo caso, per avere una migliore interpolazione, è possibile aumentare la molteplicità del punto di ascissa 0, cosicché la funzione si "avvicini" maggiormente al punto $(0, 1)$, oppure aumentare il numero dei punti di controllo.

5.3 Nodi di Chebyshev

Un'ulteriore soluzione al fenomeno di Runge proposta dallo stato dell'arte è quella definita dal matematico e statistico russo Chebyshev. Per limitare le oscillazioni generato dall'utilizzo dei polinomi interpolanti su nodi equidistanti, egli propose l'uso di nodi così definiti:

$$x_k = \cos\Big(\frac{2k+1}{n+1}\frac{\pi}{2}\Big). \tag{5.12}$$

Questi nodi sono anche definiti come zeri del polinomio di Chebyshev. In figura 8 è possibile osservare l'interpolazione polinomiale con i nodi di Chebyshev. Come è possibile notare, non è presente il fenomeno di Runge nonostante non siano stati utilizzati i polinomi di Bernstein.

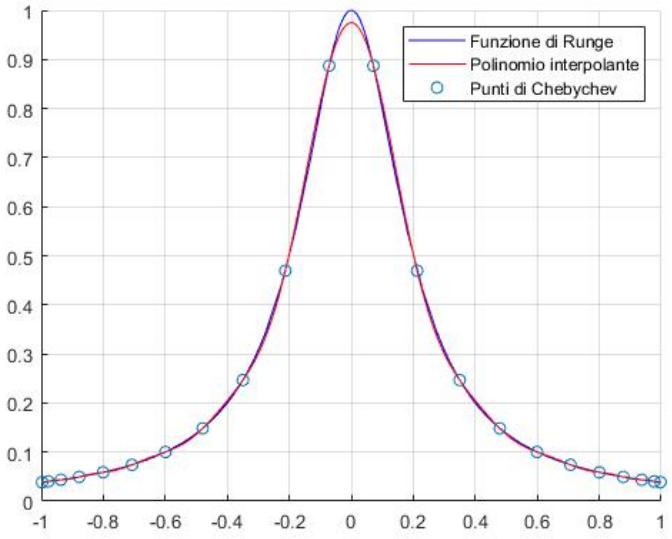


Figura 8: Interpolazione coi nodi di Chebyshev

6 Funzioni polinomiali interpolanti a tratti

Una delle maggiori difficoltà nell'utilizzo delle curve di Bézier è legata al fatto che tali curve sono "globali"; infatti, una volta definite le funzioni di base, alterando un punto viene modificata tutta la curva. Per risolvere tale problematica si definiscono delle funzioni che permettano di apportare delle modifiche locali, come le funzioni polinomiali interpolanti a tratti.

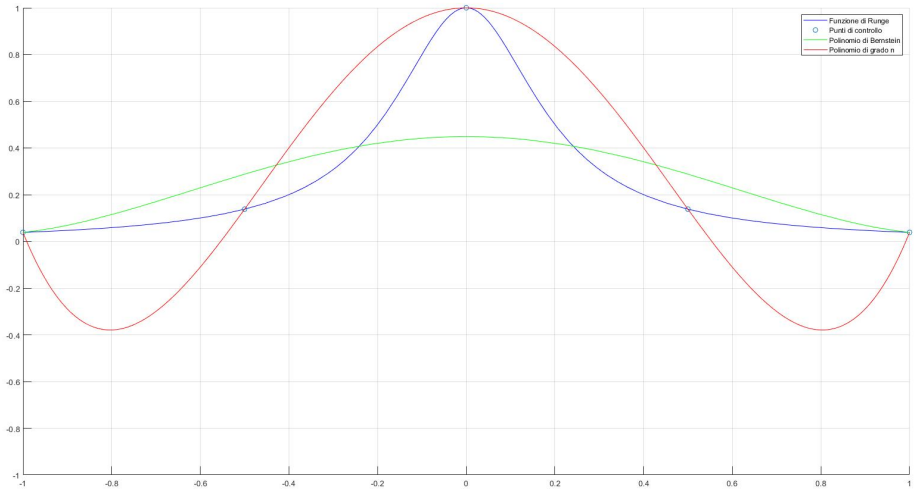
Definizione 6.1 (**Funzione polinomiale e a tratti**)**.** Sia dato un intervallo $[a, b]$ e sia $a = t_0 < t_1 < \ldots < t_n = b$. Una funzione f si dice polinomiale a tratti se la sua restrizione ad ogni intervallo $[t_k, t_{k+1}]$, con $k = 0, \ldots, n-1$, è un polinomio.

Definizione 6.2 (**Interpolante polinomiale a tratti di grado s**)**.** Siano $x_0 = a < t_1 < \ldots < t_n = b$ e si supponga che sia

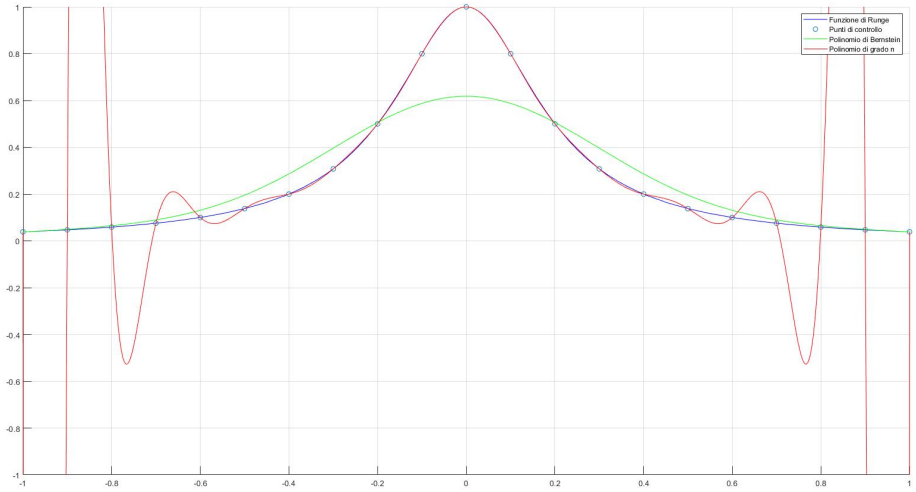
$$t_k = x_{k\cdot s} < x_{k\cdot s+1} < \ldots < x_{k\cdot(s+1)-1} < x_{k\cdot(s+1)} = t_{k+1}.$$

Si consideri la funzione $p_s^*(t)$ tale che in ogni sottointervallo $[t_k, t_{k+1}]$, $k = 0, \ldots, n$, sia il polinomio di grado s che interpola i punti $x_{ks}, \ldots, x_{(k+1)s}$ con ordinata $y_{ks}, \ldots, y_{(k+1)s}$. Tale $p_s^*(t)$ si chiama funzione polinomiale a tratti di grado s, interpolante le coppie $(x_j, y_j)_{j=0,\ldots,n}$.

In generale, i punti di raccordo $x_s, x_{2s}, \ldots$ sono punti angolosi della funzione interpolante e questa è una proprietà non desiderabile che viene superata dalle funzioni Spline.
In figura 10 è rappresentata la funzione di Runge interpolata a tratti con polinomi di grado 1 e 3.

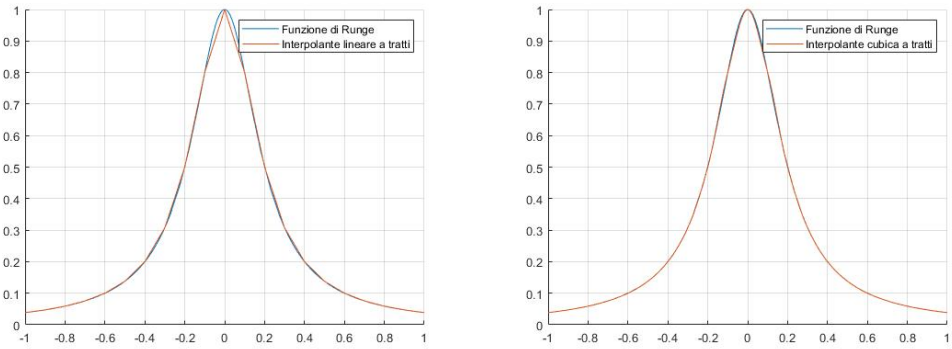


(a) Interpolazione con 5 punti di controllo



(b) Interpolazione con 21 punti di controllo

Figura 9: Interpolazione della funzione di Runge. In blu la funzione di Runge. In Rosso polinomio interpolante di grado #punti di controllo - 1. In verde interpolazione coi polinomi di Bernstein.



(a) Interpolazione lineare a tratti (b) Interpolazione cubica a tratti

Figura 10: Interpolazione polinomiale a tratti della funzione di Runge

7 Funzioni Spline

Le funzioni che permettono di costruire un modello interpolante ottimale che presenta polinomi di grado moderatamente basso e che abbia una regolarità sufficiente dell'intera curva risultante sono le **Funzioni Spline**. Il termine inglese *spline* significa *listello di legno*; ciò deriva dal fatto che originariamente la spline costituiva uno strumento di disegno formato da lunghe fettucce elastiche fissate ai nodi dell'interpolazione da grossi pesi che consentiva di tracciare curve regolari, molto utile per i profili di aerei, navi, etc.

Sia $a = x_0 < x_1 < \ldots < x_n = b$ una suddivisione dell'intervallo [a,b]. Una funzione *spline* di grado p con nodi $\{x_i\}_{i=0,1,\ldots,n}$ è una funzione indicata con $s_p(x)$ tale che, nell'intervallo [a,b] si ha:

- in ogni sottointervallo di nodi $[x_i, x_{i+1}]$ con $i = 0, 1, \ldots, n-1$, la funzione $s_p(x)$ è un polinomio di grado p.
- $s_p(x)$ e le sue prime $p-1$ derivate sono continue.

Indicando con $s_{pj}(x)$ la restrizione della spline sul sottointervallo $[x_j, x_{j+1}]$, per $j = 0, \ldots, n-1$, si può pensare a $s_{pj}(x)$ nella forma

$$s_{pj}(x) = \sum_{i=0}^{p} a_{ij}(x - x_j)^i. \tag{7.1}$$

Il numero dei coefficienti a_{ij} da determinare sono $n(p+1)$. Attraverso le condizioni di continuità nei nodi interni

$$s^{(k)}_{p,j-1}(x_j) = s^{(k)}_{pj}(x_j) \qquad j = 1, \ldots, n-1, \quad k = 0, \ldots, p-1, \tag{7.2}$$

si determinano $p(n-1)$ equazioni. Imponendo il passaggio della spline per i punti $\{(x_i, y_i)\}_{i=0,\ldots,n}$ rimangono da determinare $p-1$ coefficienti. Altre condizioni che si possono imporre affinché si ottenga un'unica soluzione del sistema, sono:

$$s^{(k)}_p(a) = s^{(k)}_p(b) \qquad k = 1, \ldots, p-1, \tag{7.3}$$

le spline che soddisfano tali condizioni sono dette *spline periodiche*;

$$s_p^{(m+j)}(a) = s_p^{(m+j)}(b) = 0 \qquad j = 0, \ldots, m-2 \tag{7.4}$$

le spline che soddisfano tali equazioni sono dette *spline naturali*, purché $p = 2m-1$ con $m \geq 2$.

La spline, dunque, mantiene la continuità delle derivate fino all'ordine $p-1$ nei punti di raccordo. Questo vuol dire che l'ultimo punto di una curva ed il primo punto della curva successiva coincidono e che la tangente e la curvatura delle due curve è la stessa. Infatti in quel punto le due curve risultano identiche poiché i valori della derivata prima e della derivata seconda coincidono.

7.1 Spline cubica interpolante

Dato un insieme di nodi $K = \{x_0 < x_1 < \ldots < x_n\}$ e un insieme di valori $\{y_i\}_{i=0,\ldots,n}$, determinare la spline cubica $s_3(x)$ tale che $s_3(x_i) = y_i$ con $i = 0, \ldots, n$ significa determinare in ogni intervallo $[x_i, x_{i+1}]$ il polinomio di III grado $s_{3,i}(x)$ che rappresenta la spline, ossia determinare i suoi coefficienti in una base di rappresentazione. Si hanno dunque 4 coefficienti per ogni intervallo per un totale di $4n$ incognite da ricavare. Per quanto riguarda le condizioni sulle funzioni, si hanno:

- $n+1$ condizioni di interpolazione $s_3(x_i) = y_i$
- 3(n-1) condizioni di continuità

$$s_{3,i-1}^k(x_i) = s_{3,i}^k(x_i) \qquad i = 1, \ldots, n-1; \quad k = 0, 1, 2. \tag{7.5}$$

Il numero totale di condizioni è dunque $n+1+3(n-1) = 4n-2$ che non coincide con $4n$ poiché mancano due condizioni per determinare univocamente le spline. Per risolvere questo problema mal posto si andranno ad aggiungere due *condizioni al contorno*, chiamate così perché vengono imposte nei nodi esterni:

$$s_3^2(x_0) = s_3^2(x_n) = 0 \tag{7.6}$$

Imponendo queste condizioni si ottiene la spline cubica naturale, che risulta essere la migliore funzione interpolante in quanto minimizza le oscillazioni tra i nodi ed è la più utilizzata nelle applicazioni. La spline cubica al crescere del numero di nodi migliora, convergendo al fenomeno che ha originato i dati. Tale risultato è dimostrato anche matematicamente: sia $y_i = g(x_i)$ con $g \in C^4[x_0, x_n]$ allora

$$R_n = |g(x) - s_{3,n}(x)| \leq \left(\frac{h^4}{384}\right) \max(g^4(x)) = O(h^4), \qquad h = \max |x_{i+1} - x_i|.$$

Quindi R_n diminuisce all'aumentare di n e

$$\lim_{n \to +\infty} s_{3,n}(x) = g(x).$$

Quindi vi è convergenza. Relazioni analoghe si hanno per le derivate fino alla III, ottenendo una buona approssimazione di $g(x)$ e delle sue derivate.

7.2 Algoritmo per il calcolo della spline cubica interpolante

L'algoritmo è formato dai seguenti passi:

1. Si costruisce la matrice A e il vettore dei termini noti b;
2. Si risolve il sistema $AM = b$, dove M è il vettore degli $n + 1$ nodi con $M_0 = 0$ e $M_n = 0$, con l'algoritmo di Gauss senza pivoting;
3. Si determina l'intervallo $[x_i, x_{i+1}]$ a cui appartiene z;
4. Si calcolano i quattro coefficienti del polinomio che rappresenta la spline in $[x_i, x_{i+1}]$;
5. Si valuta la spline z.

La complessità dell'algoritmo è $\mathcal{O}(n)$. Se la distanza tra i nodi non è troppo grande o troppo piccola il problema è ben condizionato.

8 B-Spline

8.1 Funzioni B-Spline

Le B-Spline sono funzioni spline a supporto compatto linearmente indipendenti così definite:

$$B_{i,k}(t) \qquad i = 0, \ldots, n, \tag{8.1}$$

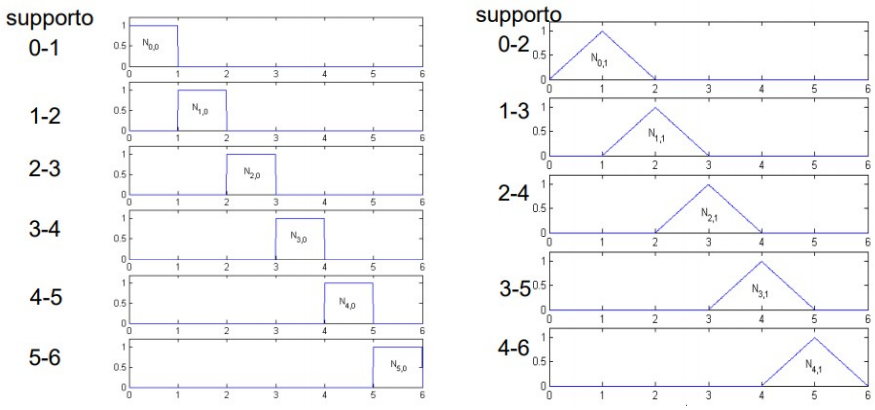
dove k è il grado del polinomio ed i il nodo considerando. Al variare del grado del polinomio, il supporto compatto è così caratterizzato:

- $B_{i,0}$ ha supporto compatto in $[t_i, t_{i+1})$;
- $B_{i,1}$ ha supporto compatto in $[t_i, t_{i+2})$;
- ...
- $B_{i,k}$ ha supporto compatto in $[t_i, t_{i+1+k})$.

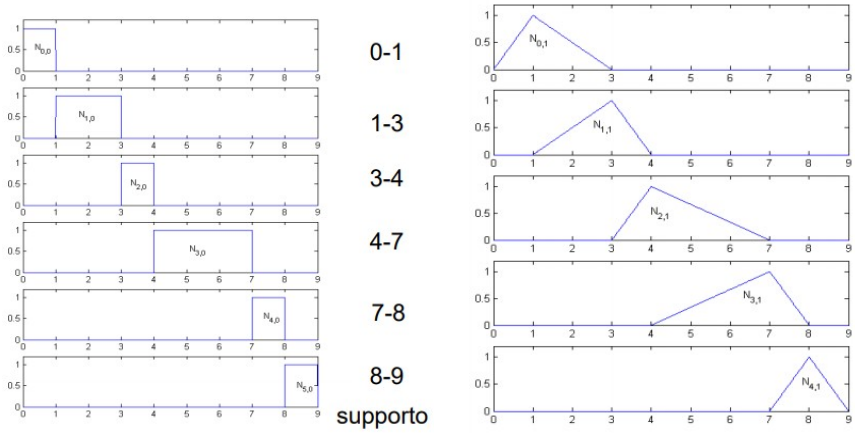
Le B-Spline si suddividono in 2 diverse categorie come mostrato in figura 11:

- B-Spline uniformi: gli intervalli hanno la stessa dimensione;
- B-Spline non uniformi: gli intervalli hanno dimensioni differenti.

La differenza sostanziale tra le Spline e le B-Spline sta nel fatto che quest'ultime possono essere funzioni diverse in base all'intervallo, mentre le spline hanno l'obbligo di essere identiche nei vari sottointervalli. Ciò permette un migliore interpolazione dei dati che si stanno rappresentando.



(a) B-Spline uniformi



(b) B-Spline non uniformi

Figura 11: B-spline di grado 0 e di grado 1

8.2 Curve B-Spline

Dati $n+1$ punti di controllo $P = \{P_i\}_{i=0,\dots,n}$ e un vettore di $m+1$ nodi $T = [t_0, t_1, \dots, t_m]$, la curva B-Spline $C(t)$ definita da P e T è:

$$C(t) = \sum_{i=0}^{n} B_{i,k}(t) \cdot P_i, \tag{8.2}$$

Come accade per le curve di Bézier, le curve B-Spline sono ottenute come combinazioni lineare dei punti di controllo P_i e le funzioni B-Spline. Queste possono essere calcolate con la formula ricorsiva di **de Boor**:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x) \qquad k > 0. \tag{8.3}$$

	Bézier	**B-SPLINE**
Contenuta nel guscio convesso del poligono di controllo?	Sì	Sì
Intervallo t?	$[0, 1]$	$[a, b]$
Grado?	Numero vertici poligono - 1	Indipendente dal poligono di controllo
Passa per i punti del poligono di controllo?	Solo per il primo e l'ultimo	In generale no, ma collassando tanti nodi quanto l'ordine della curva, può passare per qualsiasi punto del poligono
La forma della curva rispecchia quella del poligono di controllo?	No, solo approssimativamente	Sì. Con particolari accorgimenti può anche coincidere col poligono di controllo
Trasformazione affine sulla curva?	Basta operare sul poligono di controllo	Basta operare sul poligono di controllo
Controllo locale?	No: se si modifica un punto del poligono di controllo, cambia tutta la curva	Sì: se si modifica un punto del poligono di controllo, cambia solo il tratto di curva relativo
Si possono disegnare circonferenze?	No	Sì, con curve razionali

Tabella 1: Confronto tra le curve di Bézier e le B-Spline [7]

La funzione base della ricorsione è:

$$B_{i,0}(x) = \begin{cases} 1 & \forall x \in [t_i, t_{i+1}) \\ 0 & \text{altrove} \end{cases}. \tag{8.4}$$

Se per costruire la curva di Bézier servono $n + 1$ punti, ovvero i punti di controllo, per le curve B-Spline servono più informazioni.
Le funzioni spline sono definite a partire dai nodi T, che è una sequenza di numeri reali non decrescente (non è detto che debba essere strettamente crescente: alcuni nodi possono coincidere). La cardinalità dell'insieme T dipende dal numero dei punti di controllo e dal grado delle funzioni B-Spline, infatti

$$m = n + p + 1, \tag{8.5}$$

ovvero

$$\#nodi = \#punti + p + 1. \tag{8.6}$$

In altri termini, il numero dei nodi è dato da dal numero dei punti di controllo sommato all'ordine delle funzione B-Spline.
In generale, la curva spline non passa per i punti di controllo, nemmeno per il primo e l'ultimo punto a differenza della curva di Bézier. Affinché passi per questi, è possibile aumentare la molteplicità di un nodo. Se la molteplicità di questo nodo è pari a $p + 1$, la curva passerà per quel dato punto di controllo. Modificando la molteplicità dei nodi, il poligono di controllo rimane invariato ma viene modificata la forma della curva generata.
In tabella 1 sono riportate le differenza tra le curve di Bézier e le curve B-Spline.

9 Curve NURBS

Essendo le curve B-Spline e di Bézier realizzate attraverso dei polinomi, non è possibile rappresentare senza errori le curve goniometriche, anche utilizzando gli sviluppi in serie di Taylor poiché l'ambiente di calcolo è un ambiente finito.
Per questo motivo sono state introdotte le **NURBS**, *Non Uniform Rational Basis-Splines*. Questo

tipo di curve sono molto utilizzate in ambienti come CAD e sono una generalizzazione delle curve B-Spline e di Bézier.
Con le NURBS è possibile rappresentare precisamente qualunque forma, sia in 2D che in 3D, con quattro informazioni principali: grado, punti di controllo, nodi e regola di stima. Ad ogni punto di controllo è associato un peso w_i che rappresenta la sua capacità di "attrarre la curva". Quando i punti di controllo hanno lo stesso peso, la curva viene definita *non razionale*, razionale altrimenti. L'equazione che rappresenta in maniera univoca una NURBS è data da:

$$C(u) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,p}(u)}{\sum_{i=0}^{n} w_i N_{i,p}(u)} \tag{9.1}$$

Come è possibile osservare, sono ottenute come rapporto di curva razionali B-Spline.
In realtà è possibile dare una definizione più sintetica delle curve di NURBS:

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u) P_i, \tag{9.2}$$

dove

$$R_{i,p}(u) = \frac{N_{i,p}(u) \cdot w_i}{\sum_{j=0}^{n} N_{j,p}(u) \cdot w_j}. \tag{9.3}$$

$N_{i,p}(u)$ sono le funzioni base B-Spline di grado p con $N_{i,0}(u)$ definito in (8.4) e la generica k-esima funzione base B-Spline è calcolata con la formula ricorsiva (8.3).
Sia $U = [u_0, u_1, \ldots, u_m]$ il vettore dei nodi. Se una sequenza di nodi comincia con un nodo con molteplicità piena, ovvero ha una molteplicità pari all'ordine della curva, termina con un nodo a molteplicità piena e tutti i nodi sono equidistanziati, allora i suoi nodi sono uniformi. Le lettere NU in NURBS stanno per *non uniformi*, indicando che i nodi di una curva NURBS possono essere non uniformi.
Se il vettore dei nodi è costituito come descritto prima, ovvero col il primo e l'ultimo nodo a molteplicità piena, la curva è una curva di Bézier: interpola il primo e l'ultimo punto ed è tangente al primo e all'ultimo segmento del poligono di controllo.

Una superficie di NURBS è così definita:

$$S(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} P_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} N_{i,p}(u) N_{j,q}(v)}, \tag{9.4}$$

dove $w_{i,j}$ sono i pesi dati ai punti di controllo $P_{i,j}$, $N_{i,p}(u)$ e $N_{j,q}(v)$ sono le B-splines normalizzate rispettivamente di grado p e q. I vettori dei nodi sono così definiti:

$$U = [0, 0, \ldots, 0, u_{p+1}, \ldots, u_{r-p-1}, 1, 1, \ldots, 1], \tag{9.5}$$

$$V = [0, 0, \ldots, 0, v_{q+1}, \ldots, v_{s-q-1}, 1, 1, \ldots, 1], \tag{9.6}$$

dove i nodi sono ripetuti con molteplicità $p+1$ e $q+1$, con $r = n+p+1$ e $s = m+q+1$.

9.1 Proprietà delle NURBS

Non è facile comprendere le più profondi ragioni per cui le NURBS sono utilizzate in ambienti come CAD e computer grafica, tuttavia in questo paragrafo si descrivono alcune principali proprietà di cui godono le NURBS senza tralasciarne le criticità e la difficoltà che il loro impiego comporta.

Alcuni principali motivi del loro successo sono:

- Offrono una forma matematica comune per rappresentare e progettare sia forme analitiche standard (coniche, quadriche, superfici di rivoluzione...) sia curve e superfici in forma libera.
- La valutazione è ragionevolmente veloce e computazionalmente stabile.
- Hanno chiare interpretazioni geometriche e ciò le rende particolarmente utili per i progettisti che hanno una buona conoscenza della geometria descrittiva.
- Sono invarianti rispetto al ridimensionamento, alla rotazione, alla traslazione, al taglio e alla proiezione parallela e prospettica.
- NURBS dispone di un potente kit di strumenti geometrici che può essere utilizzato per progettare, analizzare ed elaborare gli oggetti.
- Sono una naturale generalizzazione delle B-spline e delle curve e superfici di Bézier razionali e non razionali.

Oltre a queste vantaggiose caratteristiche che portano le NURBS ad essere impiegate in settori importanti, esse presentano delle caratteristiche non sempre convenienti:

- Vi è la necessità di conservare più informazioni anche per curve e superfici *tradizionali*. Per esempio, per rappresentare una circonferenza usando un quadrato circoscritto servono 7 punti di controllo e 10 nodi. In una rappresentazione tradizionale basterebbero il centro, il raggio e il vettore normale al piano contenente la circonferenza. In grafica 3D bisognerebbe utilizzare 38 punti.
- Alcune *interrogazioni* sono più facili con le tecniche tradizionali piuttosto che con le NURBS. Per esempio è più difficile individuare l'intersezione tra due superfici, trovare punti di tangenza...
- Uno sbagliato uso dei pesi dei punti di controllo può portare ad una cattiva parametrizzazione.

Le proprietà di cui godono le curve NURBS sono:

- *Generalizzazione*: se $w_i = 1 \ \forall i$, allora

$$R_{i,p}(u) = \begin{cases} B_{i,p}(u) & \text{se } U = [0, 0, \ldots, 0, 1, 1, \ldots, 1] \\ N_{i,p}(u) & \text{altrimenti} \end{cases} \tag{9.7}$$

  dove gli zeri e gli uni sono ripetuti con molteplicità $p + 1$ e $B_{i,p}(u)$ rappresentano i polinomi di Bernstein di grado p.

- *Località*: $R_{i,p}(u) = 0$ se $u \notin [u_i, u_{i+p+1})$

Tipo	Funzioni	NP = poli	GR curva	Peso H
Bézier	polinomiali	utente	GR = NP - 1	No
B-Spline	polinomiali	utente	utente	No
NURBS	razionali	utente	utente	utente

Tabella 2: Confronto su alcuni parametri delle curve di Bézier, B-Spline e NURBS

	Bézier	**B-Spline**	**NURBS**
La curva è contenuta all'interno dell'involucro convesso del poligono di controllo	Sì	Sì	Sì, se i pesi associati ai punti del poligono di controllo sono maggiori di 0
Legame tra il grado della curva e il poligono di controllo	grado = numero di vertici - 1	Indipendente dal poligono di controllo	Indipendente dal poligono di controllo
La curva tocca il poligono di controllo	Sì, nel primo e nell'ultimo punto	Non è detto. Per farlo, si fanno collassare tanti nodi quanto l'ordine della curva	Per farlo, si fanno collassare tanti nodi quanto l'ordine della curva
La forma della curva e del poligono sono in qualche modo correlate?	No, solo in modo molto approssimato	Si e possono coincidere con particolari accorgimenti	Sì e possono coincidere
Controllo locale sulla curva	No	Sì	Sì

Tabella 3: Confronto tra le curve di Bézier, B-Spline e NURBS

- *Partizionamento dell'unità*:
$$\sum_i R_{i,p} = 1 \tag{9.8}$$
- *Differenziabilità*: tra i nodi, la curva appartiene a C^∞. Sui nodi, la curva appartiene a C^{p-k}. Ciò comporta che all'aumentare della molteplicità di un nodo, decresce il livello di continuità; inversamente, all'aumentare del grado p, quest'ultimo aumenta.
- $R_{i,p}(u; w_i = 0) = 0$
- $R_{i,p}(u; w_i \to +\infty) = 1$
- $R_{i,p}(u; w_j \to +\infty) = 0 \qquad j \neq i$

Di seguito sono illustrate le conseguenze delle proprietà sopra elencata.

- *Local approximation*: se un punto di controllo viene spostato o ne viene modificato il peso, il numero di intervalli della curva che vengono affetti da questa modifica sono $p + 1$; in tal modo, non viene modificata tutta la curva, come accade per le curve di Bézier.
- *Strong convex hull*: se $u \notin [u_i, u_{i+1})$, allora, la curva $C(u)$ giace all'interno dell'inviluppo convesso di $P_{i-p}, \ldots, P_i$.
- Settando il peso di un punto di controllo a zero, tale punto non ha alcun effetto sull'intera curva;
- Se $w_i \to +\infty$, allora:
$$C(u) = \begin{cases} P_i & \text{se } u \in (u_i, u_{i+p+1}) \\ C(u) & \text{altrimenti} \end{cases}. \tag{9.9}$$

In tabella 2 e in tabella 3 sono riassunte le principali differenze delle precedentemente discusse curve di Bézier, B-Spline e NURBS.

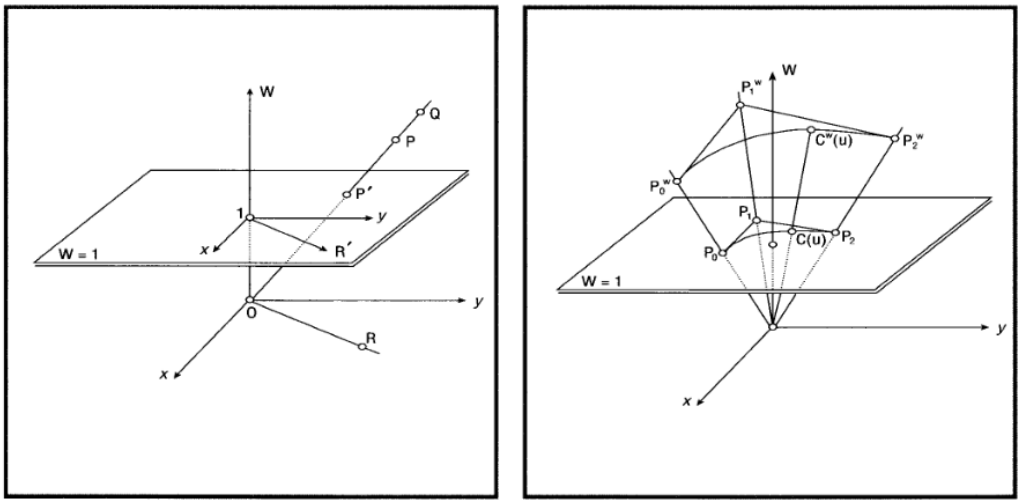


Figura 12: Costruzione geometrica di una curva NURBS [5]

9.1.1 Invarianza alle trasformazioni affini

Come affermato precedentemente, le curve NURBS sono invarianti rispetto al ridimensionamento, alla rotazione, alla traslazione, al taglio e alla proiezione parallela e prospettica. Applicando una trasformazione affine alla curva, si ottiene:

$$A[C(u)] = L[C(u)] + T = \sum_i L[P_i]R_{i,p}(u) + T. \tag{9.10}$$

D'altra parte, si ha:

$$\sum_i A[P_i]R_{i,p}(u) = \sum_i (L[P_i] + T)R_{i,p}(u) = \sum_i L[P_i]R_{i,p}(u) + T\sum_i R_{i,p}(u) = \sum_i L[P_i]R_{i,p}(u) + T, \tag{9.11}$$

per il *Partizionamento dell'unità* (9.8).

9.2 Coordinate omogenee

Prima di discutere del significato geometrico delle NURBS e comprendere come ottenere la definizione di NURBS in uno spazio euclideo di dimensione n a partire da una B-Spline in uno spazio euclideo $n+1$, vi è la necessità di analizzare il concetto di coordinate omogenee.
Si consideri uno spazio euclideo in 3D avente gli assi X, Y, W. In questo spazio, si consideri un piano di equazione $W = 1$ su cui è stato definito un'ulteriore sistema di coordinate x e y, con $x \parallel X$ e $y \parallel Y$.
Si consideri un punto P' su questo piano. Il segmento OP' individua una retta. Le coordinate (XP, YP, WP) dei punti P di questa retta vengono chiamate **coordinate omogenee**. Naturalmente, anche le coordinate di $Q \neq P$ facente parte della stessa retta sono chiamate coordinate omogenee; questo vuol dire che il segmento OP' individua una classe di coordinate omogenee tutte proporzionali tra di loro. Ovviamente, nonostante P possa essere scelto in maniera arbitraria, P

non può assumere valore (0,0,0) poiché coinciderebbe con l'origine e la retta generata dal segmento OP degenererebbe in un punto.
Ci sono 2 casi particolari da dover tenere in conto:

- $P = P'$: in questo caso $P' = (x, y, 1)$ dove $x = X/W$ e $y = Y/W$.
- Se il punto $R = (X, Y, 0)$ allora il segmento R non interseca il piano x,y. Allora il punto R' sul piano sarà rappresentato da una direzione ed è chiamato *punto all'infinito*.

A questo punto è possibile mappare i punti 3D sul piano x, y:

$$\varphi(X, Y, W) = \begin{cases} \left(\frac{X}{W}, \frac{Y}{W}\right) & W \neq 0 \\ direction(X, Y) & W = 0 \end{cases}. \tag{9.12}$$

Adesso è possibile comprendere come i pesi w_i associati ai punti di controllo P_i influenzino la curva dal punto di vista geometrico.

1. Si definiscono i punti di controllo pesati:

$$P_i^w = (w_i \cdot x_i, w_i \cdot y_i, w_i) \quad i = 0, \ldots, n. \tag{9.13}$$

2. Si costruisce la curva B-Spline non razionale nello spazio col sistema di coordinate X, Y, W:

$$C^w(u) = \sum_{i=0}^{m} P_i^w N_{i,p}(u). \tag{9.14}$$

3. Si mappa la curva sul piano x, y:

$$C(u) = \varphi(C^w(u)) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,p}(u)}{\sum_{i=0}^{n} w_i N_{i,p}(u)}. \tag{9.15}$$

Da quest'ultima analisi è possibile ribadire ancora una volta che se $w_i = 1 \quad \forall i$, la curva NURBS degenera in una B-Spline; questo perché il denominatore della (9.15) è pari ad 1 e i punti di controllo P_i coincidono coi i punti P_i' del piano.
Come naturale conseguenza di quanto appena esposto, è possibile dare una definizione di una NURBS tridimensionale, facilmente generalizzabile anche ad altre dimensioni: una NURBS tridimensionale è una proiezione di una curva B-Spline quadrimensionale.

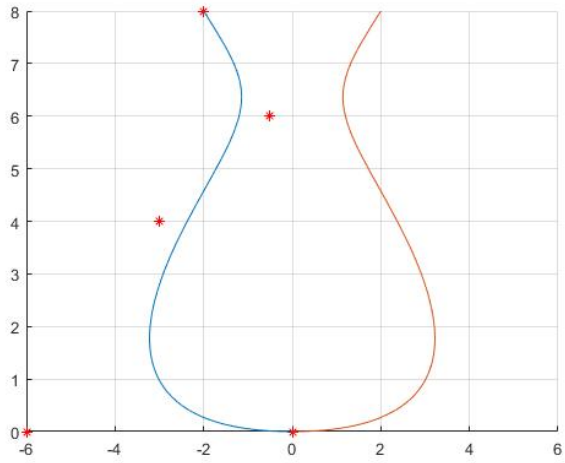


Figura 13: Profilo di un vaso. Con * sono indicati i punti di controllo. La curva blu è la curva di Bézier. La curva arancione è ottenuta riflettendo la curva di Bézier sull'asse $x = 0$.

10 MATLAB

In questo paragrafo si riportano i codici MATLAB degli script sviluppati per mettere in pratica i concetti esposti in questo elaborato.

10.1 Profilo di un vaso in 2D con le curve di Bézier

```
clear all; close all; clc;

% Punti di controllo
P =[0  0; -6  0; -3  4; -0.5  6; -0.5  6;  -2  8];

% Generazione della matrice di Bernstein
syms t
B = bernsteinMatrix(length(P(:,1)) - 1, t);
curve = simplify(B*P);

% Plot dei punti di controllo e della curva generata
figure
hold on
grid on
xlim([-6  6])
plot(P(:,1), P(:, 2), '*r')
fplot(curve(1), curve(2), [0  1])
fplot(-curve(1), curve(2), [0  1])
```

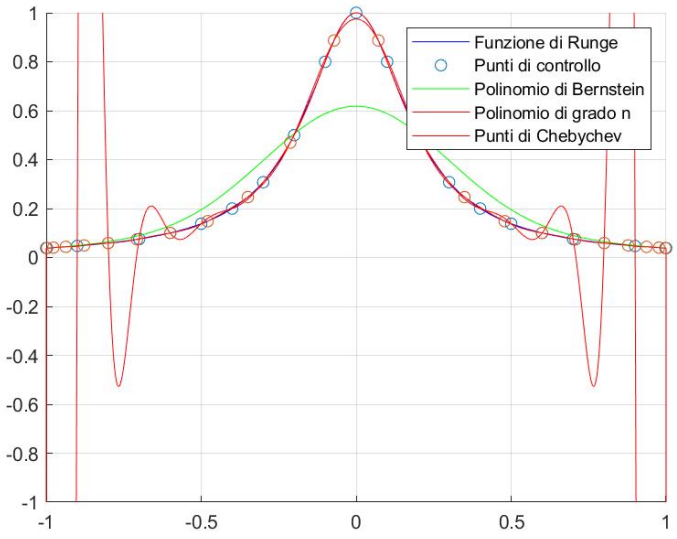


Figura 14: Interpolazione della funzione di Runge con il polinomio di Bernstein, con un polinomio di grado n, ed utilizzando i nodi di Chebyshev

10.2 Interpolazione della funzione di Runge

```
clear all; close all; clc;

R = @(x) 1./(1+25.*x.^2); % Funzione di Rounge

% Funzione di Rounge corretta
x = -1:0.1:1;
y = R(x);

% Polinomio di Bernstein
P = [x' y'];              % Matrice di punti
syms t
B = bernsteinMatrix(length(x) - 1, t);
curve = simplify(B*P);  % Curva

% Polinomio interpolante di grando length(x) - 1
p = polyfit(x, y, length(x) - 1);
s = linspace(-1,1, 1000);
f = polyval(p,s);

% Chebychev
n = length(x);
```

```
k = 0:1:n;
cx = cos((2*k+1)/(n+1) * pi/2);
cy = R(cx);
pc = polyfit(cx, cy, n);
fc = polyval(pc, s);

% Plot delle figure
figure()
grid on
hold on
ylim([-1,1])
fplot(R, [-1,1], 'b')                    % Plot di Rounge tra -1 e 1
plot(x,y,'o')                            % Plot dei punti di controllo
fplot(curve(1), curve(2), [0 1], 'g');   % Plot del polinomio di
    Bernstein
plot(s,f, 'r');                          % Plot del polinomio
    interpolante
plot(s,fc, 'r');                         % Plot Chebychev
plot(cx, cy, 'o')
legend('Funzione di Runge', 'Punti di controllo', 'Polinomio di
    Bernstein', 'Polinomio di grado n', 'Punti di Chebychev')
```

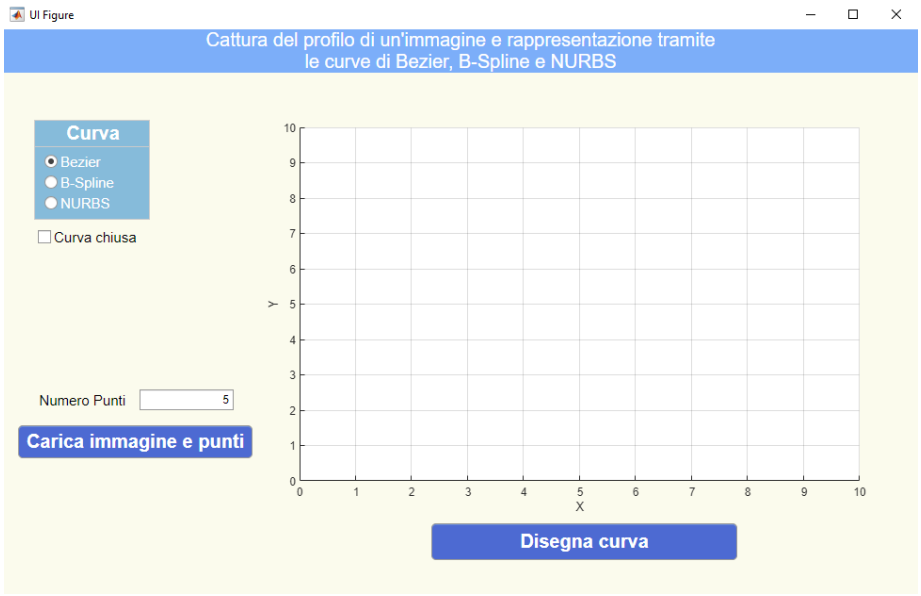


Figura 15: Interfaccia grafica dell'applicativo

10.3 Applicazione per la rappresentazione del profilo di un oggetto

In questo paragrafo vi è la descrizione dell'applicazione realizzata tramite *MATLAB App Designer*. L'applicazione permette ad un utente di poter realizzare le curve di Bézier, B-Spline e NURBS (chiuse o aperte) definendo quanti e quali punti di controllo utilizzare settandoli su una immagine da lui scelta.

10.3.1 Interfaccia

In figura 15 è riportata l'interfaccia all'apertura dell'applicazione. Essa presenta in alto a sinistra un *Radio Button Group*, per la scelta del tipo di curva desiderata, e una *Check Box* per selezionare il tipo di curva, se chiusa o aperta. In basso a sinistra è presente un *Edit Number Field* tramite il quale è possibile stabilire il numero di punti di controllo desiderati. Inoltre, è presente il *Button* **Carica immagine e punti** che consente di selezionare l'immagine desiderata e stabilire i punti di controllo sull'immagine stessa.
In basso al centro è presente un ulteriore *Button*, **Disegna curva**, che se cliccato, una volta selezionati i punti di controllo, mostra sul grafico la curva scelta.

10.3.2 Funzionamento

La prima operazione da effettuare è scegliere il tipo di curva e l'opzione di curva chiusa o aperta (può essere modificato anche in seguito). Successivamente bisogna definire il numero di punti di controllo e cliccare il bottone *Carica immagine e punti* per poter scegliere l'immagine desiderata. Come mostrato in figura 16, infatti, si aprirà l'esplora risorse del rispettivo sistema operativo che permetterà di selezionare l'immagine.

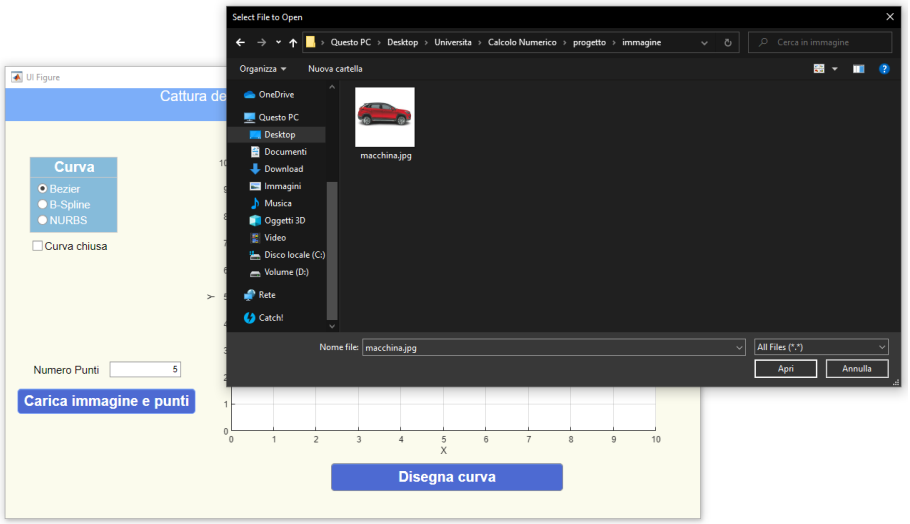


Figura 16: Selezione dell'immagine di cui si vuole disegnare il profilo

Dopodiché si aprirà un'ulteriore schermata nella quale è presente l'immagine scelta con la possibilità di definire un numero di punti di controllo stabiliti in precedenza. La schermata si chiuderà automaticamente una volta impostati e basterà semplicemente cliccare sul bottone *Disegna curva* per mostrarla a video.
È possibile modificare il valore nel *Radio Button Group* e della *Check Box*, senza dover riacquisire i punti di controllo, e cliccare nuovamente su *Disegna Curva*.

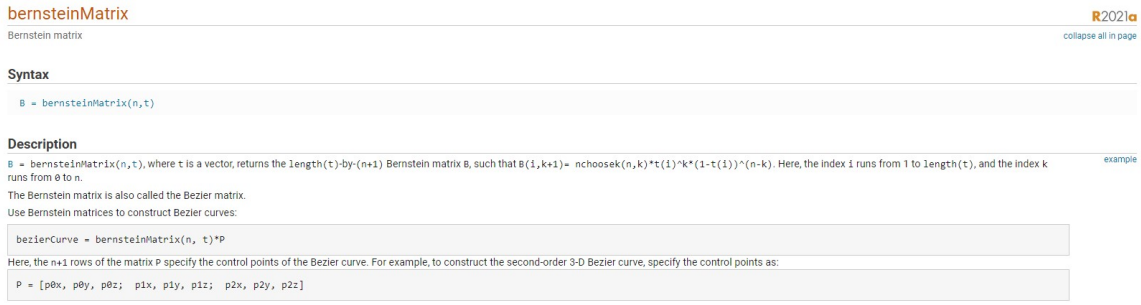
10.3.3 Implementazione in MATLAB

Di seguito sono riportate le varie funzioni di *Callback* per i due buttons presenti nell'applicazione.

```
function DisegnacurvaButtonPushed(app, event)

            global isRead;
            if isRead == false
                return
            end

            hold(app.UIAxes, 'off')
            global imm;
            imshow(imm, [], 'parent', app.UIAxes);
            hold(app.UIAxes, 'on')

            global xp;
            global yp;

```

```
16                  if  app.CurvachiusaCheckBox.Value == true
17                      xp1 = [xp; xp(1)];
18                      yp1 = [yp; yp(1)];
19                  else
20                      xp1 = xp;
21                      yp1 = yp;
22                  end
23
24
25                  if app.BezierButton.Value == true
26                      syms t
27                      B = bernsteinMatrix(length(xp1) - 1, t);
28                      P = [xp1 yp1];
29                      curve = simplify(B*P);
30                      fplot(app.UIAxes, curve(1), curve(2), [0 1], 'b', '
                            LineWidth', 4);
31                      plot(app.UIAxes, xp1, yp1, '-xy', 'LineWidth', 3);
32
33                  else if app.BSplineButton_2.Value == true
34                      cpts = [xp1'; yp1'];
35                      tpts = [0 5];
36                      tvec = 0:0.01:5;
37                      [q, ~, ~,pp] = bsplinepolytraj(cpts, tpts, tvec);
38                      plot(app.UIAxes, cpts(1,:), cpts(2,:), '-xy', '
                            LineWidth', 3);
39                      [point, t] = fnplt(pp);
40                      plot(app.UIAxes, point(1,:), point(2,:), 'g', '
                            LineWidth', 4);
41                  else
42                     P = [xp1'; yp1'];
43                     n_punti = length(xp1);
44                     n = 3;
45                     t = [0 0 0:1/(n_punti - n + 1):1 1 1];
46                     w = ones(1,n_punti);
47                     C = nurbsfun(n,t,w,P);
48                     plot(app.UIAxes, C(1,:), C(2,:), 'k', 'LineWidth', 4);
49                     plot(app.UIAxes, xp1, yp1, '-xy', 'LineWidth', 3);
50
51                  end
52
53  end
```

La prima function riportata è *DisegnacurvaButtonPushed*, collegata al button *Disegna Curva*. All'inizio della funzione viene effettuato un controllo per vedere se l'immagine è stata caricata e per mostrarla a video nella parte *Grafico* dell'interfaccia.



bernsteinMatrix

R2021a

Bernstein matrix

collapse all in page

Syntax

```
B = bernsteinMatrix(n,t)
```

Description

example

`B = bernsteinMatrix(n,t)`, where `t` is a vector, returns the `length(t)`-by-`(n+1)` Bernstein matrix `B`, such that `B(i,k+1)= nchoosek(n,k)*t(i)^k*(1-t(i))^(n-k)`. Here, the index `i` runs from `1` to `length(t)`, and the index `k` runs from `0` to `n`.

The Bernstein matrix is also called the Bezier matrix.

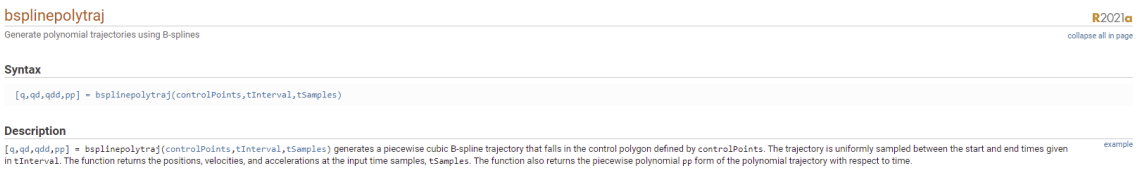
Use Bernstein matrices to construct Bezier curves:

```
bezierCurve = bernsteinMatrix(n, t)*P
```

Here, the `n+1` rows of the matrix `P` specify the control points of the Bezier curve. For example, to construct the second-order 3-D Bezier curve, specify the control points as:

```
P = [p0x, p0y, p0z;  p1x, p1y, p1z;  p2x, p2y, p2z]
```

Figura 17: Documentazione MATLAB della funzione *bernsteinMatrix*



bsplinepolytraj

R2021a

Generate polynomial trajectories using B-splines

collapse all in page

Syntax

```
[q,qd,qdd,pp] = bsplinepolytraj(controlPoints,tInterval,tSamples)
```

Description

example

`[q,qd,qdd,pp] = bsplinepolytraj(controlPoints,tInterval,tSamples)` generates a piecewise cubic B-spline trajectory that falls in the control polygon defined by `controlPoints`. The trajectory is uniformly sampled between the start and end times given in `tInterval`. The function returns the positions, velocities, and accelerations at the input time samples, `tSamples`. The function also returns the piecewise polynomial `pp` form of the polynomial trajectory with respect to time.

Figura 18: Documentazione MATLAB della funzione *bsplinepolytraj*

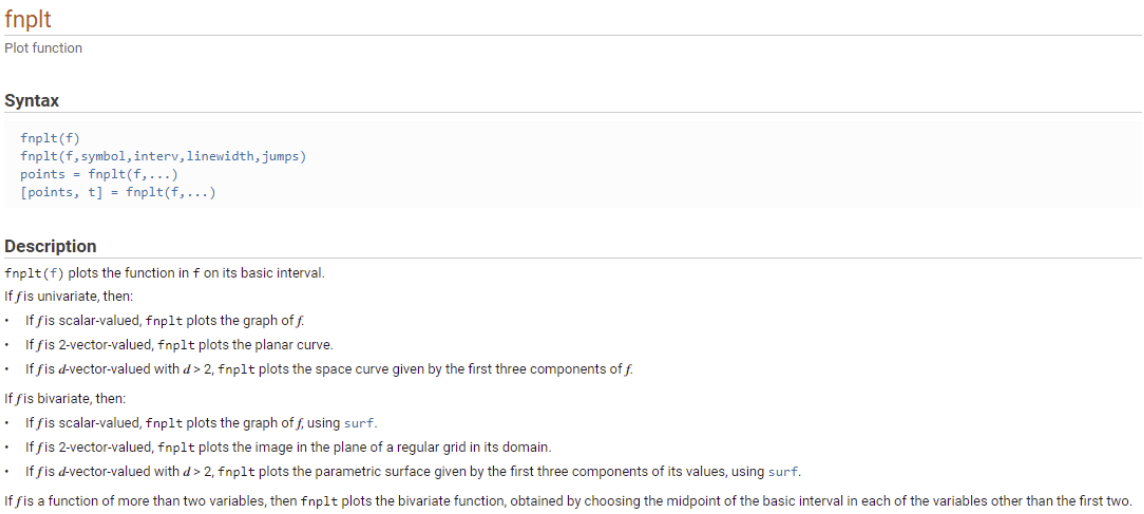
Successivamente viene fatto un controllo sul valore della *Check Box* per vedere se la curva da disegnare debba essere chiusa o aperta. Nel caso in cui la curva fosse chiusa, viene aggiunto un ulteriore punto di controllo finale che coincide con quello iniziale.
In seguito viene eseguito un controllo per valutare il valore del *Radio Button Group*.

Bézier Viene utilizzata la funzione *bernsteinMatrix*, la cui documentazione è riportata in figura 17, tramite la quale si ottiene la matrice dei polinomi di Bernstein.
Viene poi definita la matrice dei punti di controllo ed applicata la funzione *simplify*, che semplifica ogni elemento del prodotto tra la matrice dei punti di controllo e la matrice dei polinomi di Bernstein, ottenendo la curva che successivamente viene plottata tramite *fplot*. L'ultimo comando all'interno dell'if serve per plottare anche i punti di controllo e con "-xy" è possibile disegnare il segmento che li unisce a due a due.

B-Spline All'inizio vengono definiti i punti di controllo, gli estremi dell'intervallo temporale e i campioni temporali per la traiettoria, necessari per la funzione *bsplinepolytraj*, della quale in figura 18 è riportata la documentazione. In seguito, tramite *fnplt* (documentazione in figura 19), funzione presente nel *Curve Fitting Toolbox*, si ottengono i punti che verranno successivamente plottati per ottenere la B-Spline risultante.

NURBS All'inizio vengono definiti i punti di controllo, l'ordine delle curve B-Spline, il vettore dei nodi, la cui cardinalità è data dalla (8.5), e il vettore dei pesi in funzione del numero dei punti di controllo scelto. Successivamente viene utilizzata una funzione esterna chiamata *nurbsfun* che consente di ottenere la curva NURBS che viene plottata tramite la linea di codice successiva.

La seconda funzione riportata è *CaricaimmagineepuntiButtonPushed* collegata al button *Carica immagine e punti*.



fnplt

Plot function

Syntax

```
fnplt(f)
fnplt(f,symbol,interv,linewidth,jumps)
points = fnplt(f,...)
[points, t] = fnplt(f,...)
```

Description

`fnplt(f)` plots the function in `f` on its basic interval.

If *f* is univariate, then:

- If *f* is scalar-valued, `fnplt` plots the graph of *f*.
- If *f* is 2-vector-valued, `fnplt` plots the planar curve.
- If *f* is *d*-vector-valued with *d* > 2, `fnplt` plots the space curve given by the first three components of *f*.

If *f* is bivariate, then:

- If *f* is scalar-valued, `fnplt` plots the graph of *f*, using `surf`.
- If *f* is 2-vector-valued, `fnplt` plots the image in the plane of a regular grid in its domain.
- If *f* is *d*-vector-valued with *d* > 2, `fnplt` plots the parametric surface given by the first three components of its values, using `surf`.

If *f* is a function of more than two variables, then `fnplt` plots the bivariate function, obtained by choosing the midpoint of the basic interval in each of the variables other than the first two.

Figura 19: Documentazione MATLAB della funzione *fnplt*

```
function CaricaimmagineepuntiButtonPushed(app, event)

        global imm;
        global isRead;
        [name, path] = uigetfile;
        if name == 0
            isRead = false;
            return
        end
        isRead = true;
        imm = imread(strcat(path, name));
        [dimx, dimy] = size(imm(:,:,1));
        app.UIAxes.XLim = [0 dimx];
        app.UIAxes.YLim = [0 dimy];
        imshow(imm, [], 'parent', app.UIAxes);
        hold(app.UIAxes, 'on')
        figure(1);
        imshow(imm, []);
        grid on;
        hold on;
        n_punti = app.NumeroPuntiEditField.Value;
        global xp;
        global yp;
        try
            [xp, yp] = ginput(n_punti);
        catch ignore
```

```
27                % CHIUSURA FIGURE CON X
28                return
29            end
30            close(1)
```

La prima operazione effettuata è la richiesta del path dell'immagine che viene successivamente letta tramite il comando *imread*.
In seguito, vengono salvate le dimensioni di quest'ultima in modo tale da poter definire gli assi della figura per poi mostrarla a video nella sezione *Grafico* dell'interfaccia.
Infine, vengono selezionati i punti di controllo tramite il comando *ginput*, che consente di identificare le coordinate dei punti selezionati con input da mouse.

10.3.4 Esempio

Di seguito è mostrato un test dell'applicazione variando il numero di punti di controllo e il tipo di curva.
È stata presa un immagine di un'auto e l'obiettivo è quello di tracciare la curva che si avvicina maggiormente alla forma della carrozzeria.
Nelle immagini in figura 21 e in figura 22 sono mostrati i risultati delle prove effettuate con 20 e 40 punti di controllo, sia selezionando la curva chiusa che aperta, anche se quella più di interesse per questo elaborato è la curva chiusa.

10.3.5 Considerazioni

Come è possibile apprezzare dalle immagini, la curva di Bézier è quella che si comporta peggio; inoltre, all'aumentare del numero di punti, tende addirittura a rappresentare delle linee che si allontanano dal risultato atteso. Nonostante ciò, le curve di Bézier sono comunque molto utilizzate nell'ambito della modellazione grafica 2D e 3D in programmi di computer graphics come Blender. In figura 20 è riportato un esempio del suo utilizzo per la creazione di un vaso in 3D.
Per quanto riguarda i risultati ottenuti con le curve B-Spline e le NURBS è possibile notare che con esse si ottiene un'ottima approssimazione dell'immagine iniziale e che vi è un netto miglioramento aumentando i punti di controllo. Infatti, utilizzando 40 punti di controllo è stato possibile avere un buon contorno della carrozzeria dell'automobile.

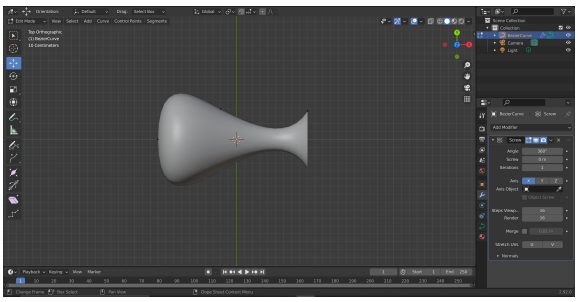
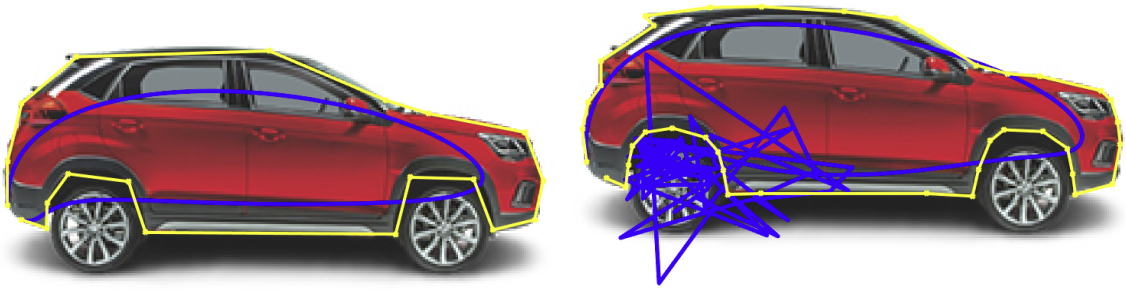


Figura 20: Modellazione di un vaso in Blender

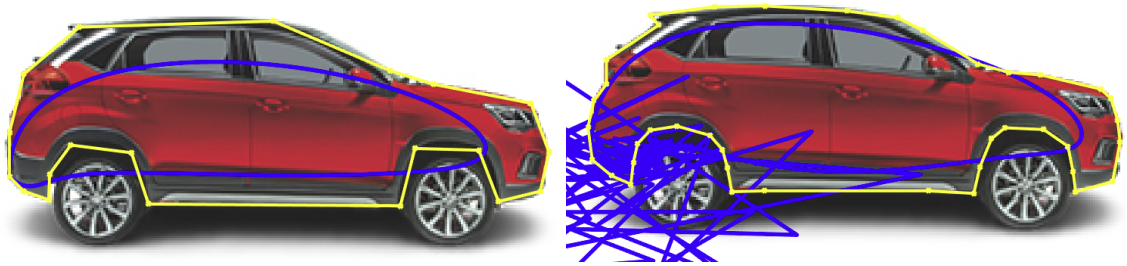


(a) Profilo con curva di Bézier

(b) Profilo con curva B-Spline

(c) Profilo con curva NURBS

Figura 21: Profilo di un'autovettura dati a sinistra 20 punti di controllo, a destra 40, con differenti curve aperte



(a) Profilo con curva di Bézier

(b) Profilo con curva B-Spline

(c) Profilo con curva NURBS

Figura 22: Profilo di un'autovettura dati a sinistra 20 punti di controllo, a destra 40, con differenti curve chiuse

Riferimenti bibliografici

[1] Dario A. Bini, *Il problema dell'interpolazione*, Università di Pisa, http://pagine.dm.unipi.it/bini/Didattica/AnaNum/testi/Dispense/interpolazione.pdf

[2] Dario A. Bini, *V. Pan Polynomial and Matrix Computations*, Birkhäuser, 1994

[3] Ghelardoni Paolo, *Interpolazione e approssimazione*, Dispense https://people.dm.unipi.it/ghelardoni/libro/interpolazione.pdf

[4] Inglese G., Gautschi W., *Lower Bounds for the Condition Number of Vandermonde Matrices*, Numer. Math. 52, 241-250, 1988

[5] Les Piegl, *On NURBS: A Survey*, University of South Florida, 1991

[6] Sommariva Alvise, *Interpolazione polinomiale a tratti e spline*, Università degli Studi di Padova, Dispensa 2019

[7] Università degli studi di Milano, *Curve B-Splines*, 13/06/2021 http://www.mat.unimi.it/users/alzati/Geometria_Computazionale_98-99/apps/bezspli/bsplines.html