



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Relazione
di
Metodi Numerici per la Grafica

Di
Federico Schipani

A.A. 2017-2018

Indice

1 La Base delle B-Spline	2
1.1 Esempi di basi	4
1.2 Proprietà della base delle B-Spline	4
2 Curve B-Spline	7
2.1 Proprietà delle curve B-Spline	7
2.2 Continuità	13
2.3 Curve Chiuse	17
2.4 Interpolazione	18
3 Superfici	20
3.1 Superfici parametriche	20
3.2 Superfici <i>Tensor-Product</i> di Bézier	20
3.2.1 Proprietà delle superfici di Bézier	23
3.2.2 Algoritmo di De Casteljau	26

Elenco delle figure

1 Base con $\mathbf{t} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ e $k = 5$	4
2 Base con nodi multipli di ordine 6	5
3 Base di Bernstein di ordine 5	5
4 Supporto locale	6
5 Dettaglio della prima splines $N_{1,4}(t)$ per $t = [0.85, 1.15]$	7
6 Partizione dell'unità	8
7 Trasformazione affine su spline	9
8 Proprietà di località	10
10 Proprietà di Variation Diminishing	13
11 Curve discontinue	14
12 Continuità C^0	15
13 Continuità C^1	16
14 Continuità G^1	16
15 Curva chiusa (in rosso i vertici ripetuti)	18
16 Interpolazione di punti	19
17 Superficie parametrica	21
18 Base di Bézier	22
19 Superficie di Bézier generata da Codice 11	23
20 Trasformazione affine su una superficie di Bézier	25
21 Curve di bordo	26

22	Superfici disegnate con <i>De Casteljau</i>	29
----	---	----

Listings

1	Calcolo delle basi di Cox De Boor	3
2	Applicazione trasformazione affine	9
3	Proprietà di località	10
4	Proprietà di località	11
5	Proprietà di Variation Diminishing	12
6	Continuità sulle curve	13
7	Curva chiusa	17
8	Interpolazione	19
9	Disegno di una superficie parametrica	20
10	Base delle superfici di Bézier	21
11	Disegno di una superficie di Bézier	22
12	Trasformazione affine	24
13	Curve di bordo	25
14	Algoritmo di De Casteljau	26

1 La Base delle B-Spline

Dato un vettore esteso dei nodi

$$\mathbf{t} = \left\{ \underbrace{t_0, \dots, t_{k-2}}_{k-1}, \underbrace{t_{k-1}, \dots, t_{n+1}}_{\tau_0, \tau_1, \dots, \tau_L}, \underbrace{t_{n+2}, \dots, t_{n+k}}_{k-1} \right\}$$

con

$$\mathbf{t}_0 \leq t_1 \leq \dots t_{k+1} < t_k \dots < t_{n+1} \leq t_{n+2} \leq \dots \leq t_{n+k}$$

possiamo definire la base delle *B-Spline* su nodi semplici tramite la relazione ricorrente di *Cox - De Boor*.

Definizione 1. Le *B-Spline* di ordine 1, oppure grado 0 sono definite come:

$$N_{i,1}(t) = \begin{cases} 1, & \text{se } t \in [t_i, t_{i+1}] \\ 0, & \text{altrimenti} \end{cases} \quad i = 0, \dots, n+k-1$$

Altrimenti le *B-Spline* di ordine $r \leq k$ sono definite ricorsivamente, per $r > 1$, come:

$$N_{i,r}(t) = \omega_{i,r}(t)N_{i,r-1}(t) + [1 - \omega_{i+1,r}(t)]N_{i+1,r-1}$$

dove

$$\omega_{i,r}(t) = \begin{cases} \frac{t-t_i}{t_{i+r-1}-t_i}, & \text{se } t < t_{i+r-1} \\ 0, & \text{altrimenti} \end{cases}$$

Le *B-Spline* possono anche essere definite su una partizione nodale la cui molteplicità m_i di un generico nodo τ_i è più alta di 1, quindi su nodi multipli. In questo caso il vettore esteso dei nodi diventa:

$$\mathbf{t} = \left\{ \underbrace{t_0, \dots, t_{k-2}}_{k-1}, \underbrace{t_{k-1}, \dots, t_{n+1}}_{\tau_0, \tau_1, \dots, \tau_L}, \underbrace{t_{n+2}, \dots, t_{n+k}}_{k-1} \right\}$$

con τ_i ripetuto a seconda della sua molteplicità m_i con $i = 1, \dots, L-1$ in \mathbf{t} , e

$$\mathbf{t}_0 \leq t_1 \leq \dots t_{k+1} \leq t_k \dots \leq t_{n+1} \leq t_{n+2} \leq \dots \leq t_{n+k}$$

La definizione della base delle *B-Spline* di *Cox - De Boor* non cambia, ma bisogna stare attenti in quanto $\omega_{i,r}(t)$ può diventare nullo per qualche valore r a causa dei nodi multipli. In Codice 1 sono mostrate le due funzioni che calcolano le basi di *Cox - De Boor*, realizzate senza l'utilizzo delle funzioni del *Curve Fitting Toolbox*.

Codice 1: Calcolo delle basi di Cox De Boor

```

1 function [omega] = calc_omega (i, r, t_star, t)
2     if t(i) == t(i+r-1)
3         omega = 0;
4         return;
5     elseif t_star <= t(i+r-1)
6         omega = (t_star-t(i)) / (t(i+r-1)-t(i));
7         return;
8     else
9         omega = 0;
10        return;
11    end
12 end
13
14 function [y] = de_boor_basis (i, r, t, t_star, k)
15    if r == 1
16        if (t_star >= t(i) && t_star < t(i+1)) || ...
17            ((t_star >= t(i) && t_star <= t(i+1) && ...
18             t_star == t(end) && i == length(t)-k))
19
20            y = 1;
21            return;
22        else
23            y = 0;
24            return;
25        end
26    else
27        omega1 = calc_omega(i, r, t_star, t);
28        omega2 = (1 - calc_omega(i+1, r, t_star, t));
29        db1 = de_boor_basis(i, r-1, t, t_star, k);
30        db2 = de_boor_basis(i+1, r-1, t, t_star, k);
31        y = omega1 * db1 + omega2 * db2;
32        return;
33    end
34 end

```

La funzione `calc_omega` di Codice 1 si può spiegare in questo modo. Dati in input l'indice i , l'ordine r , il punto in cui si vuole calcolare la spline t_{star} ed il vettore esteso dei nodi \mathbf{t} si occupa di calcolare i valori $\omega_{i,r}(t)$. Il controllo iniziale $t(i) == t(i+r-1)$ serve a gestire il caso di nodi multipli. In questa particolare condizione possiamo trovarci a gestire casi in cui il denominatore di $\frac{t-t_i}{t_{i+1}-t_i}$ è uguale a 0; quindi $\omega_{i,r}(t)$ dev'essere posto a 0. La seconda funzione in Codice 1 è `de_boor_basis` che effettua il calcolo delle basi delle B-Spline. La condizione booleana a riga 16, 17 e 18 serve a verificare che, nel caso in cui l'ordine della spline sia 1, ci si trovi all'interno dell'intervallo $[t_i, t_{i+1}]$. Bisogna però fare attenzione al caso in cui il punto $t = t_{star}$ di $N_{i,k}(t)$ si trovi nell'ultimo intervallo. Questo ha

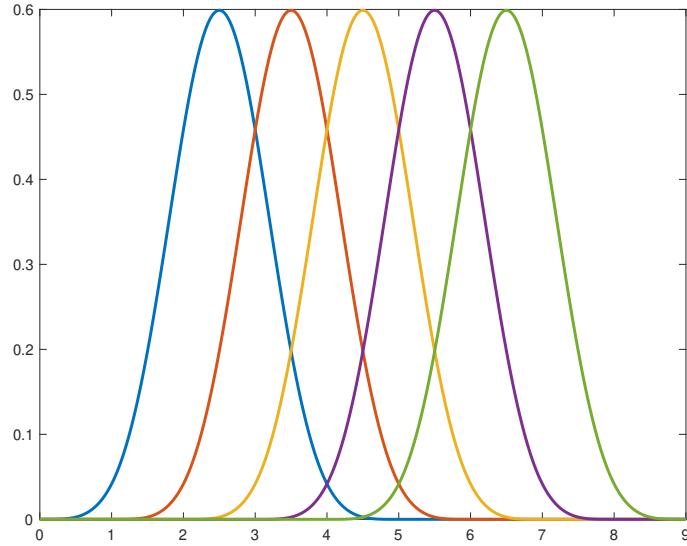


Figura 1: Base con $\mathbf{t} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ e $k = 5$

reso necessario introdurre un ulteriore controllo per fare in modo che venga preso in considerazione anche l'ultimo valore dell'ultimo intervallo.

1.1 Esempi di basi

L'esempio più immediato di base è quello dove il vettore esteso dei nodi è uniforme, in questo caso abbiamo preso $\mathbf{t} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ e $k = 5$, otterremo quindi 5 funzioni di base visualizzate in Figura 1.

Cambiando il vettore esteso dei nodi \mathbf{t} otteniamo basi per le *B-Spline* con regolarità diversa. In Figura 2 viene mostrato cosa succede tenendo fissato il numero di nodi e l'ordine e aumentato la molteplicità del nodo 4.

Un caso particolare della base delle *B-Spline* sono i polinomi di *Bernstein*. Questi ultimi si ottengono quando, dato $[a, b] = [\tau_0, \tau_L]$, la partizione nodale estesa è formata solamente da a ripetuto k volte e b ripetuto altrettante k volte. In Figura 3 è mostrato un esempio di base ottenuta con i polinomi di *Besrnstein* di grado 5.

1.2 Proprietà della base delle B-Spline

La base delle *B-Spline* gode di diverse proprietà:

1. Supporto locale: $N_{i,r}(t) = 0$ se $t \notin [t_i, t_{i+r}]$

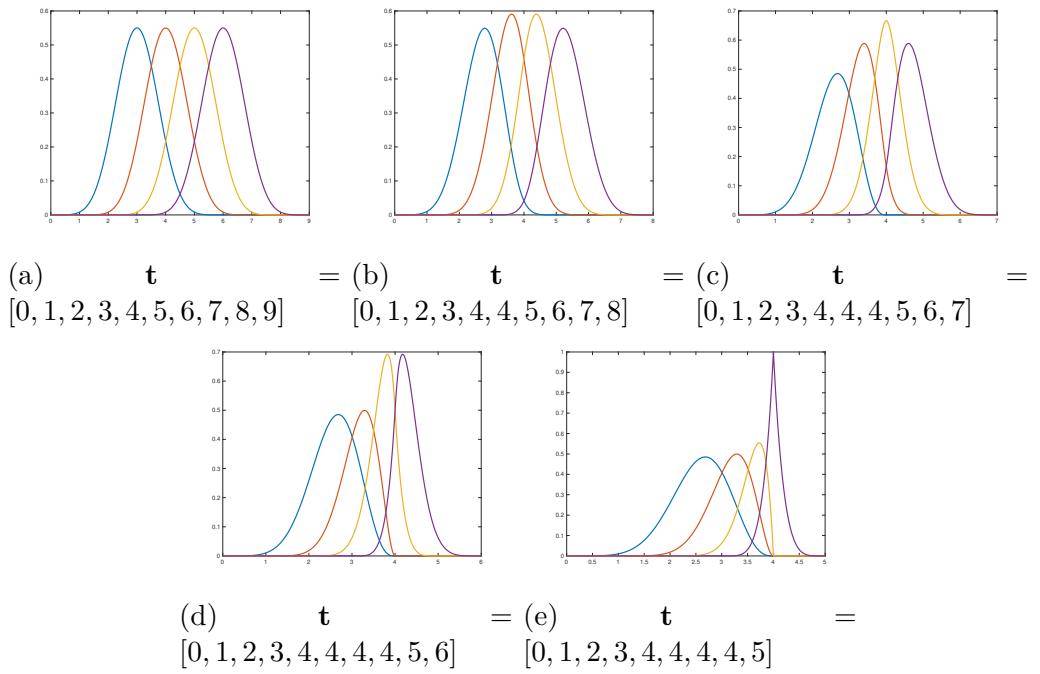


Figura 2: Base con nodi multipli di ordine 6

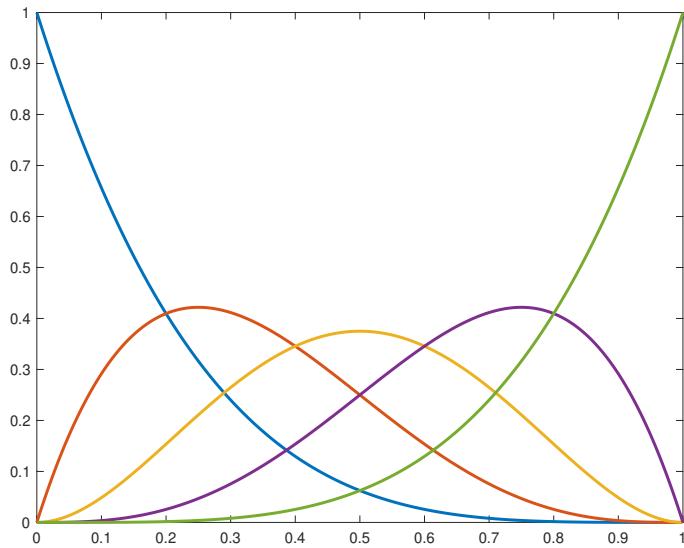


Figura 3: Base di Bernstein di ordine 5

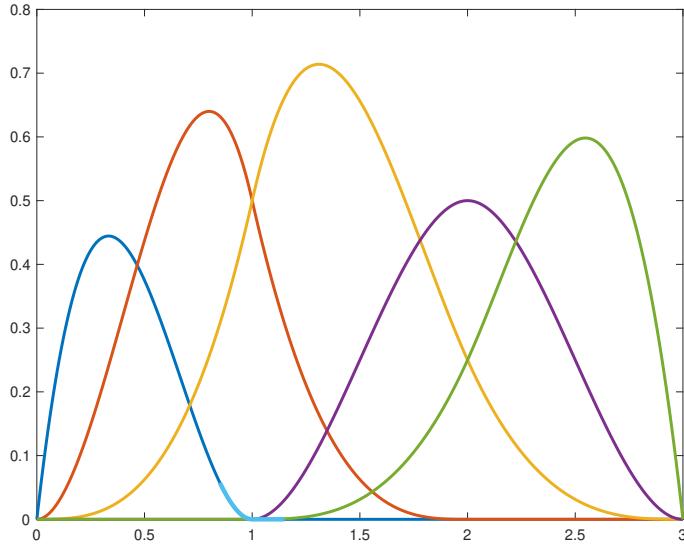


Figura 4: Supporto locale

2. Non negatività: $N_{i,r}(t) \geq 0 \forall t \in \mathbb{R}$
3. Partizione dell'unità: $\sum_{i=0}^{n+k-r} = 1 \forall t \in [t_{r-1}, t_{n+1+k-r}]$ con $r = 1 \dots k$

Supporto locale Questa proprietà ci dice che la spline $N_{i,r}(t)$ è diversa da zero solamente nell'intervallo di nodi che va da t_i a t_{i+r} . Prendiamo ad esempio le *B-Spline* di ordine $k = 4$ e con $\mathbf{t} = [0, 0, 0, 1, 1, 2, 3, 3, 3]$. Le splines saranno le seguenti:

- $N_{1,4}(t) \neq 0 t \in [t_1 = 0, t_5 = 1]$
- $N_{2,4}(t) \neq 0 t \in [t_2 = 0, t_6 = 2]$
- $N_{3,4}(t) \neq 0 t \in [t_3 = 0, t_7 = 3]$
- $N_{4,4}(t) \neq 0 t \in [t_4 = 1, t_8 = 3]$
- $N_{5,4}(t) \neq 0 t \in [t_5 = 1, t_9 = 3]$

Facendo un plot di questa base possiamo vedere come la proprietà di supporto locale sia verificata, in particolare in Figura 4 sono mostrate tutte le splines della base, mentre in Figura 5 è mostrato un dettaglio della $N_{1,4}(t)$ per $t = [0.85, 1.15]$.

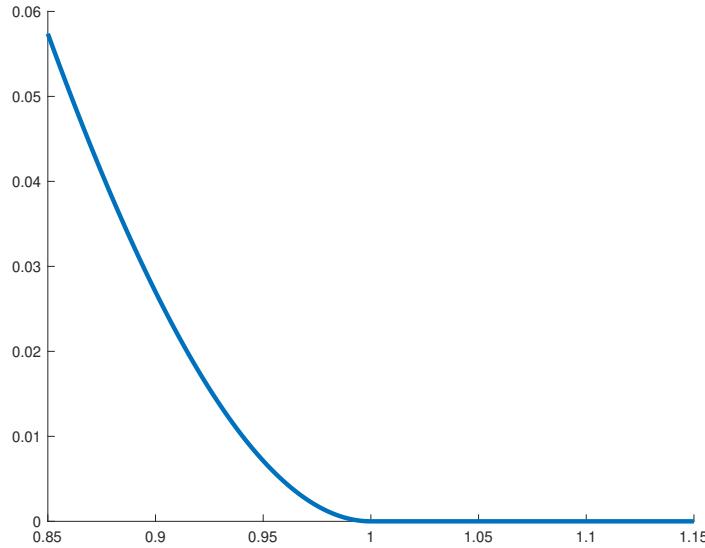


Figura 5: Dettaglio della prima splines $N_{1,4}(t)$ per $t = [0.85, 1.15]$

Non negatività In questo caso la proprietà è verificabile sfruttando uno qualsiasi dei plot mostrati in precedenza, ad esempio possiamo vedere che in Figura 4 nessuna delle $N_{i,r}(t)$ è negativa.

Partizione dell'unità Questa proprietà ci dice che la somma delle funzioni di base delle *B-Spline* definite sulla partizione nodale estesa \mathbf{t} è uguale a 1, se si considerano nell'intervallo $[\tau_0, \tau_L]$. Un esempio è mostrato in Figura 6.

2 Curve B-Spline

A partire dalla base delle *B-Spline* è possibile realizzare delle curve. Dati $n+1$ punti di controllo la curva è definita come

$$\mathbf{X}(t) := \sum_{i=0}^n \mathbf{d}_i N_{i,k}(t)$$

2.1 Proprietà delle curve B-Spline

Le curve *B-Spline* godono di diverse proprietà:

1. Invarianza per trasformazioni affini: la proprietà della base delle *B-Spline* di essere una partizione dell'unità garantisce che le curve *B-Spline* siano

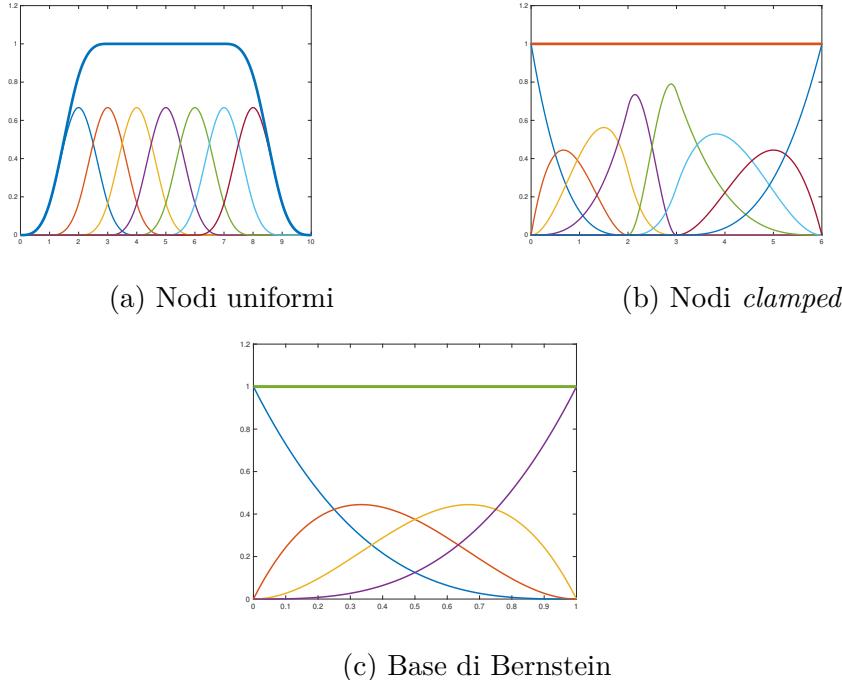


Figura 6: Partizione dell’unità

invarianti per trasformazioni affini. Questo vuol dire che l’applicazione della trasformazione affine sulla curva, o sui punti di controllo, è indifferente in quanto il risultato non cambia.

2. Località: un segmento di curva è influenzato solamente da k punti di controllo.
3. Strong Convex Hull: ogni punto sulla curva appartiene all’inviluppo convesso di k punti di controllo consecutivi, con k ordine delle funzioni spline.
4. Variation Diminishing: il numero di intersezioni tra una retta e la curva è minore o uguale al numero di intersezioni tra la stessa retta ed il poligono di controllo.

Invarianza per trasformazioni affini In Codice 2 è presente l’implementazione con cui è stata applicata una trasformazione prima ai punti di controllo e poi alla curva. Come si può vedere dal Codice 2 la trasformazione applicata è stata una rotazione di 180 gradi ed uno spostamento di 1 su entrambi gli assi. Per generare la base con cui viene disegnata la curva è stata usata la funzione `spcol` del *Curve Fitting Toolbox*. Un modo *Naïve* con cui ci si può accertare della veridicità

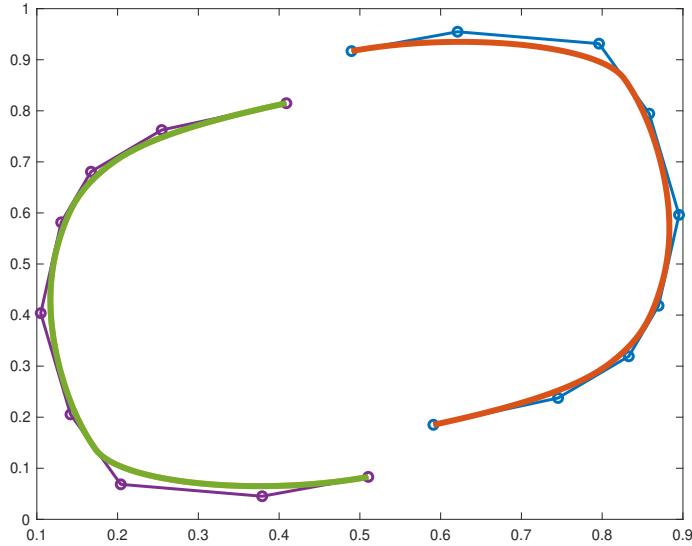


Figura 7: Trasformazione affine su spline

di questa proprietà è guardando il Codice 2 e la Figura 7. Nel codice sono presenti tre chiamate a funzione `plot`: la prima per la curva originale, la seconda per la curva sulla quale è stata applicata la trasformazione e la terza per la curva disegnata a partire dai punti di controllo sui quali è stata applicata la trasformazione. È possibile osservare che nella Figura 7 sono presenti due curve. Questo vuol dire che due curve si sono sovrapposte.

Codice 2: Applicazione trasformazione affine

```

1 k = 4;
2 knots = [0 0 0 0 1 1 2 3 4 5 5 5 5];
3 tau = knots(k):0.001:knots(end-k+1);
4 c = spcol(knots, k, tau);
5 [x_p, y_p] = ginput(length(knots)-k);
6 curve_x = zeros(size(c,1),1);
7 curve_y = zeros(size(c,1),1);
8 plot(x_p, y_p, 'o-', 'linewidth', 2); hold on;
9 for i = 1:length(x_p) %o y_p
10     curve_x = curve_x + (x_p(i) * c(:, i));
11     curve_y = curve_y + (y_p(i) * c(:, i));
12 end
13 plot(curve_x, curve_y, 'linewidth', 4); hold on;
14 %trasformazione affine sulla curva
15 theta = pi;
16 A = [cos(theta) -sin(theta) ; sin(theta) cos(theta)];

```

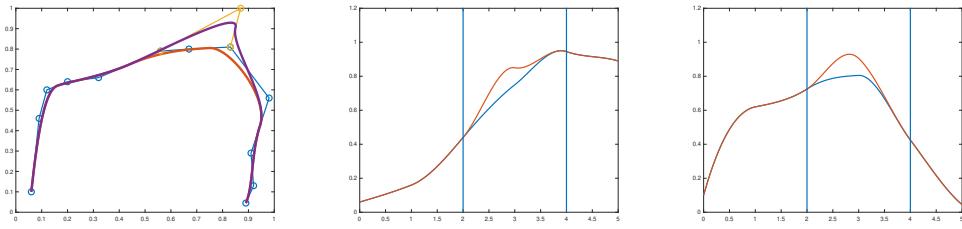
(a) Spostamento di \mathbf{d}_7 (b) Variazione sulla x (c) Variazione sulla y

Figura 8: Proprietà di località

```

17 new_curve = A*[curve_x curve_y]'+1;
18 plot(new_curve(1,:), new_curve(2,:), 'linewidth', 4);
19 %trasformazione affine
20 %sposto i PDC
21 new_points = A*[x_p y_p]'+1;
22 curve_x = zeros(size(c,1),1);
23 curve_y = zeros(size(c,1),1);
24 plot(new_points(1,:), new_points(2,:), 'o-', 'linewidth', 2);
    hold on;
25 for i = 1:length(x_p) %o y_p
26     curve_x = curve_x + (new_points(1,i) * c(:, i));
27     curve_y = curve_y + (new_points(2,i) * c(:, i));
28 end
29 plot(curve_x, curve_y, 'linewidth', 4); hold on;

```

Località La proprietà di località ci dice che il punto di controllo \mathbf{d}_j influenza la curva solamente per $t \in [t_j, t_{j+k}]$. Ad esempio, come mostrato in Figura 8 ed in Codice 3, spostando \mathbf{d}_7 la curva varia da $t \in [t_7, t_{11}]$.

Codice 3: Proprietà di località

```

1 k = 4;
2 knots = augknt([0 1 2 3 4 5], k, 2);
3 tau = knots(k):0.001:knots(end-k+1);
4 c = spcol(knots, k, tau);
5 x_p =
    [0.06;0.09;0.12;0.20;0.32;0.56;0.67;0.83;0.98;0.91;0.92;0.89];

6 y_p =
    [0.10;0.46;0.60;0.64;0.66;0.79;0.80;0.81;0.56;0.29;0.13;0.045];

7 curve_x = zeros(size(c,1),1);
8 curve_y = zeros(size(c,1),1);

```

```

9 plot(x_p, y_p, 'o-', 'linewidth', 2, 'markersize', 10); hold on
10;
10 for i = 1:length(x_p) %o y_p
11     curve_x = curve_x + (x_p(i) * c(:, i));
12     curve_y = curve_y + (y_p(i) * c(:, i));
13 end
14 plot(curve_x, curve_y, 'linewidth', 4); hold on;
15 x_p2 = x_p; y_p2 = y_p; move = 7;
16 x_p2(move) = x_p2(move)+0.2;
17 y_p2(move) = y_p2(move)+0.2;
18 curve_x2 = zeros(size(c,1),1);
19 curve_y2 = zeros(size(c,1),1);
20 plot(x_p2(move-1:move+1), y_p2(move-1:move+1), 'o-', 'linewidth'
21     , 2, 'markersize', 10); hold on;
21 for i = 1:length(x_p) %o y_p
22     curve_x2 = curve_x2 + (x_p2(i) * c(:, i));
23     curve_y2 = curve_y2 + (y_p2(i) * c(:, i));
24 end
25 plot(curve_x2, curve_y2, 'linewidth', 4); hold on;
26 figure(2);
27 plot(tau, curve_x, tau, curve_x2, 'linewidth', 2); hold on;
28 line([knots(move) knots(move)], [0 1.2], 'linewidth' , 2);
29 line([knots(move+k) knots(move+k)], [0 1.2], 'linewidth' , 2);
30 figure(3);
31 plot(tau, curve_y, tau, curve_y2, 'linewidth', 2); hold on;
32 line([knots(move) knots(move)], [0 1.2], 'linewidth' , 2);

```

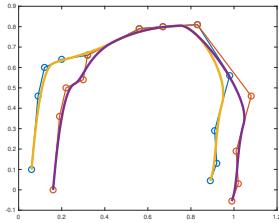
La proprietà di località ci dice anche che una curva *B-Spline* $\mathbf{X}(t^*)$ con $t^* \in [t_r, t_{r+1})$ è determinata da k punti di controllo d_{r-k+1}, \dots, d_r . Utilizzando Codice 4 viene mostrata questa proprietà. Per questo esempio è stato scelto $r = 8$, ed una partizione nodale mostrata a Riga 2 del Codice 3. I punti di controllo spostati sono 8, ovvero i $\mathbf{d}_j \notin [\mathbf{d}_5, \mathbf{d}_8]$. Come mostrato in Figura 8 la curva $\mathbf{X}(t^*)$ rimane quindi invariata per $t^* \in [t_8, t_9)$.

Codice 4: Proprietà di località

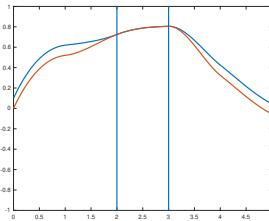
```

1 r = 8;
2 x_p3 = x_p; y_p3 = y_p;
3 x_p3(1:r-k) = x_p3(1:r-k)+0.1; y_p3(1:r-k) = y_p3(1:r-k)-0.1;
4 x_p3(r+1:end) = x_p3(r+1:end)+0.1; y_p3(r+1:end) = y_p3(r+1:end)
    -0.1;
5 curve_x2 = zeros(size(c,1),1);
6 curve_y2 = zeros(size(c,1),1);
7 for i = 1:length(x_p)
8     curve_x2 = curve_x2 + (x_p3(i) * c(:, i));
9     curve_y2 = curve_y2 + (y_p3(i) * c(:, i));
10 end
11 figure(4);

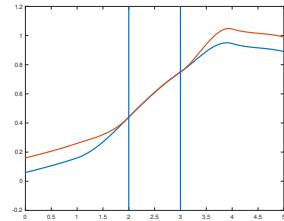
```



(a) Spostamento di $\mathbf{d}_j \notin [\mathbf{d}_5, \mathbf{d}_8]$



(b) Variazione sulla x



(c) Variazione sulla y

```

12 plot(x_p, y_p, 'o-', 'linewidth', 2, 'markersize', 10); hold on
    ;
13 plot(x_p3, y_p3, 'o-', 'linewidth', 2, 'markersize', 10)
14 plot(curve_x, curve_y, 'linewidth', 4);
15 plot(curve_x2, curve_y2, 'linewidth', 4);
16 figure(5);
17 plot(tau, curve_y, tau, curve_y2, 'linewidth', 2); hold on;
18 line([knots(r) knots(r)], [-1 1], 'linewidth', 2);
19 line([knots(r+2) knots(r+2)], [-1 1], 'linewidth', 2);
20 figure(6);
21 plot(tau, curve_x, tau, curve_x2, 'linewidth', 2); hold on;
22 line([knots(r) knots(r)], [-0.2 1.2], 'linewidth', 2);
23 line([knots(r+2) knots(r+2)], [-0.2 1.2], 'linewidth', 2);

```

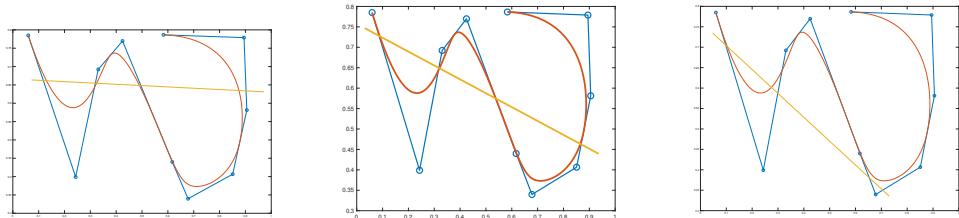
Variation Diminishing La proprietà di Variation Diminishing dice che presa una qualunque retta che interseca un numero b di volte il poligono di controllo, questa intersecherà un numero di volte $a \leq b$ la curva *B-Spline* disegnata a partire dal poligono di controllo. Un esempio realizzato con il Codice 5 è mostrato in Figura 10.

Codice 5: Proprietà di Variation Diminishing

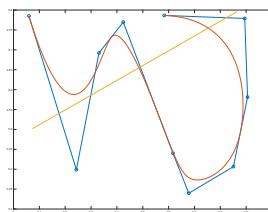
```

1 k = 4;
2 knots = [0 0 0 0 1 1 2 3 4 4 5 5 5 5];
3 tau = knots(k):0.001:knots(end-k+1);
4 c = spcol(knots, k, tau);
5 [x_p, y_p] = ginput(length(knots)-k);
6 curve_x = zeros(size(c,1),1);
7 curve_y = zeros(size(c,1),1);
8 plot(x_p, y_p, 'o-', 'linewidth', 2, 'markersize', 10); hold on
    ;
9 for i = 1:length(x_p) %o y_p
10     curve_x = curve_x + (x_p(i) * c(:, i));
11     curve_y = curve_y + (y_p(i) * c(:, i));
12 end

```



(a) Poligono: 4, Curva: 4 (b) Poligono: 4, Curva: 4 (c) Poligono: 4, Curva: 2



(d) Poligono: 4, Curva: 2

Figura 10: Proprietà di Variation Diminishing

```

13 plot(curve_x, curve_y, 'linewidth', 3); hold on;
14 [x, y] = ginput(2);
15 plot(x, y, '-.', 'linewidth', 3);

```

2.2 Continuità

Consideriamo due curve di *Bézier* (che ricordiamo essere un caso particolare di curve *B-Spline*) $C_1(u)$ e $C_2(v)$, definite rispettivamente su $u \in [u_a, u_b]$ e $v \in [v_a, v_b]$ e sui vertici di controllo $\{\mathbf{V}_{1,1}, \dots, \mathbf{V}_{n,1}\}$ e $\{\mathbf{V}_{1,2}, \dots, \mathbf{V}_{m,2}\}$. Consideriamo ora il caso in cui queste due curve si uniscono, usando come punto di raccordo l'ultimo vertice della prima curva ed il primo vertice della seconda curva. Possono verificarsi diversi casi descritti di seguito. Per generare le Figure 11, 12, 13 e 14, mostrate in seguito, è stato usato il Codice 6.

Codice 6: Continuità sulle curve

```

1 k = 4;
2 knots = [0 0 0 0 1 1 2 3 4 4 5 5 5 5];
3 tau = knots(k):0.001:knots(end-k+1);
4 c = spcol(knots, k, tau);
5 [x_p, y_p] = ginput(length(knots)-k);
6 curve_x = zeros(size(c,1),1);
7 curve_y = zeros(size(c,1),1);

```

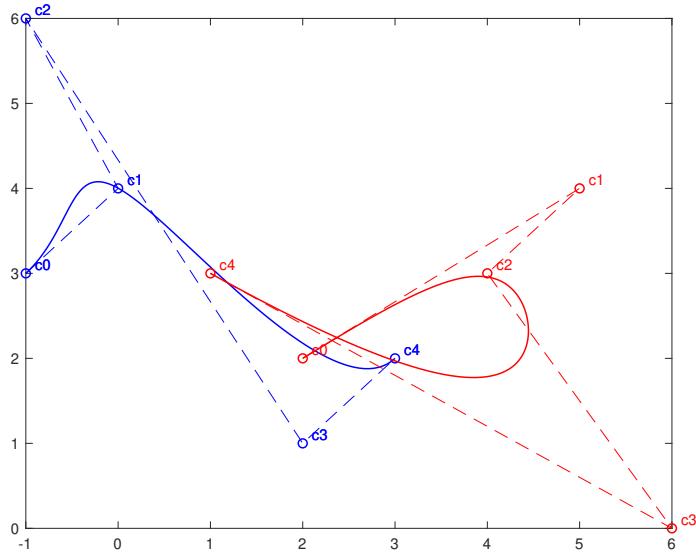


Figura 11: Curve discontinue

```

8 plot(x_p, y_p, 'o-', 'linewidth', 2, 'markersize', 10); hold on
9 ;
10 for i = 1:length(x_p) %o y_p
11     curve_x = curve_x + (x_p(i) * c(:, i));
12     curve_y = curve_y + (y_p(i) * c(:, i));
13 end
14 plot(curve_x, curve_y, 'linewidth', 3); hold on;
15 [x, y] = ginput(2);
16 plot(x, y, '-.', 'linewidth', 3);

```

Discontinuità Il primo caso è la discontinuità chiamata, con abuso di notazione, C^{-1} . Questa si verifica quando l'ultimo vertice della prima curva è diverso dal primo vertice della seconda curva, ovvero:

$$V_{n,1} \neq V_{0,2}$$

Un esempio è possibile vederlo in Figura 11.

Continuità C^0 Date due curve la condizione per far sì che ci sia continuità di tipo C^0 è che l'ultimo vertice della prima curva coincida con il primo vertice della seconda curva, ovvero che:

$$\mathbf{V}_{n,1} = \mathbf{V}_{1,2}$$

Un esempio di tale continuità è presente in Figura 12

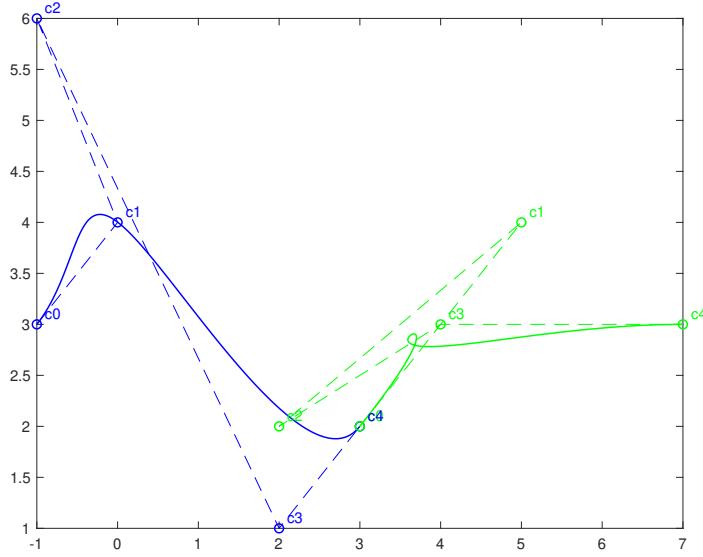


Figura 12: Continuità C^0

Continuità C^1 La continuità C^1 si impone con la seguente condizione sulla derivata prima:

$$C'_1(\mathbf{V}_{n,1}) = C'_2(\mathbf{V}_{1,2})$$

Per far sì che questa condizione sia verificata è sufficiente posizionare il penultimo vertice della prima curva, il secondo della seconda curva ed il punto di giunzione per la continuità C^0 sulla stessa retta. Analiticamente ciò vuol dire imporre la seguente ugualanza:

$$\mathbf{V}_{2,2} = 2\mathbf{V}_{n,1} - \mathbf{V}_{n-1,1}$$

Un esempio è mostrato in Figura 13.

Continuità G^1 La continuità G^1 è un particolare tipo di continuità chiamata anche *Continuità geometrica*. Questo tipo di continuità non dipende dalla parametrizzazione della curva. La condizione per far sì che sia presente questo particolare tipo di continuità è la seguente:

$$C'_2(v_0) = \omega C'_1(u_1)$$

con ω a piacere. Un esempio di continuità G^1 è mostrato in Figura 14.

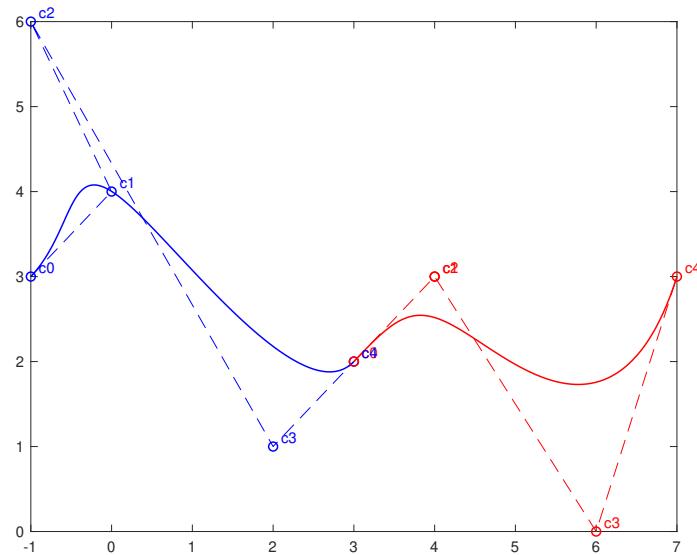


Figura 13: Continuità C^1

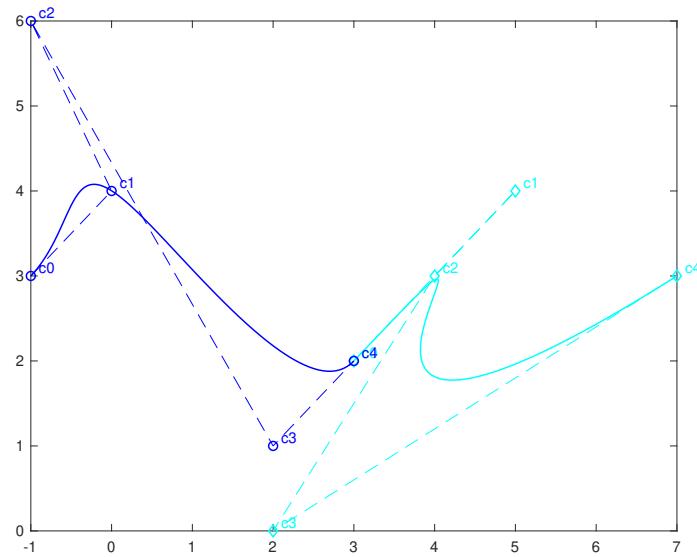


Figura 14: Continuità G^1

2.3 Curve Chiuse

Grazie ai nodi ausiliari ciclici è possibile usare la base delle *B-Spline* per generare curve chiuse. In Codice 7 è possibile vedere un esempio di implementazione di una curva chiusa di ordine $k = 4$. Per ottenere una curva con questa proprietà è necessario generare una partizione nodale estesa in questo modo:

$$\Delta^* = \left[\underbrace{\frac{-k}{m-1}}_{Inizio} : \underbrace{\frac{1}{m-1}}_{Passo} : \underbrace{\frac{k+m-1}{m-1}}_{Fine} \right]$$

con k ordine della spline e m numero di vertici di controllo della curva. Successivamente è anche necessario estendere il poligono di controllo ripetendo i primi $k - 1$ vertici.

Codice 7: Curva chiusa

```

1 k = 4;
2 x_p = [35 19 15 10 5 2 3 12 19 30 ...
3      35 19 15];
4 y_p = [6 8 8 9 9 8 5 5 5 5 ...
5      6 8 8];
6 m = (length(x_p)-k)-1;
7 knots = -k/m:1/m:(k+m)/(m);
8 plot(x_p, y_p, 'o-', 'linewidth', 2); hold on;
9 curve = spmak(knots, [x_p; y_p]);
10 fnplt(fnbrk(curve, [knots(k), knots(end-k+1)]));
11 for i = 1:k-1
12     plot(x_p(i), y_p(i), 'ro-', 'markersize', 8, ...
13           'MarkerFaceColor', 'r');
14 end
15 for i = 0:k-2
16     disp("Continuita': " + i);
17     der = fnder(curve, i);
18     disp(fnval(der, knots(k)) + " " + fnval(der, knots(end-k+1)));
19 end

```

In Figura 15 è possibile vedere il risultato del Codice 7, i vertici di controllo rossi sono quelli ripetuti. La curva chiusa generata da Codice 7 risulta continua nel punto di raccordo con continuità di tipo C^{k-1-m_i} , quindi essendo la curva di grado 3 (*e ordine 4*) fino alla derivata seconda abbiamo continuità, dalla derivata quarta in poi invece viene restituito 0.

```

Continuita': 0
"21 21"
"7.66667 7.66667"

```

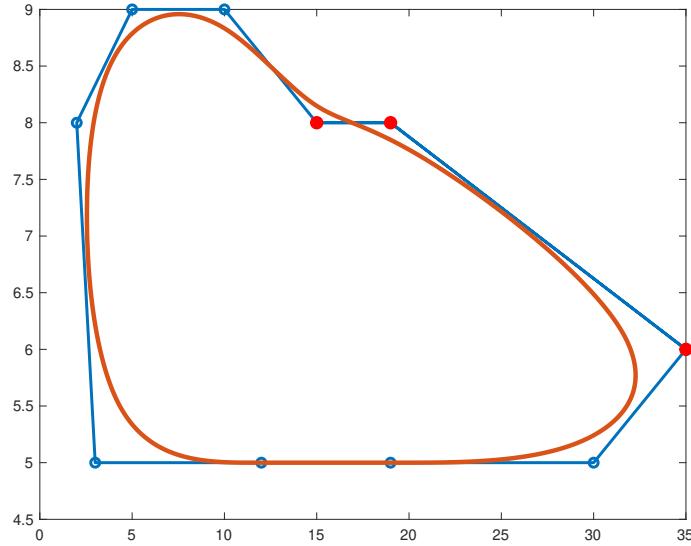


Figura 15: Curva chiusa (in rosso i vertici ripetuti)

Continuità: 1

"-80 -80"

"8 8"

Continuità: 2

"768 768"

"-128 -128"

2.4 Interpolazione

L'interpolazione consiste nel far passare la curva per un determinato insieme di punti $\mathbf{p}_1, \dots, \mathbf{p}_n$. Dopo aver scelto adeguatamente la partizione nodale è necessario generare la base per le *B-Spline*. Successivamente bisogna trovare i punti di controllo per generare la curva interpolante. Per trovare i punti di controllo della curva interpolante è necessario risolvere il seguente sistema lineare:

$$N \cdot Coefs = P$$

con N base delle *B-Spline* e P punti di interpolazione. Nella risoluzione di tale sistema bisogna stare attenti a produrre una matrice N non singolare. Il Codice 8 genera due splines interpolanti mostrate in Figura 16. I punti in rosso sono i $\mathbf{p}_1, \dots, \mathbf{p}_n$ di interpolazione, mentre rispettivamente le linee tratteggiate in blu e nero sono i punti di controllo della prima e della seconda curva interpolante.

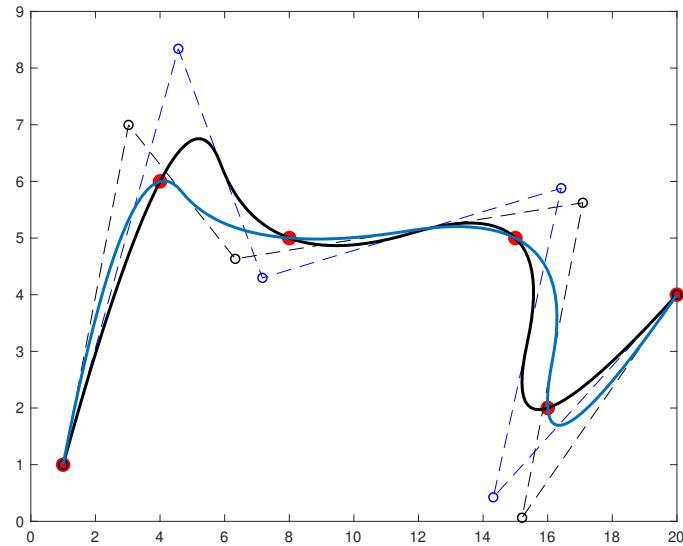


Figura 16: Interpolazione di punti

Codice 8: Interpolazione

```

1 k = 3;
2 knots = [zeros(1,k), 1/4, 2/4, 3/4, ones(1, k)];
3 t = aveknt(knots, k);
4 P = [1 1; 4 6; 8 5; 15 5; 16 2; 20 4];
5 plot(P(:,1), P(:,2), 'r.', 'markersize', 30); hold on;
6 hold on
7 A = spcol(knots, k, t);
8 coefs = A\P;
9 plot(coefs(:,1), coefs(:,2), 'b--o');
10 x = spmak(knots, coefs.');
11 fnplt(x, 'k');
12 t1 = uniform_param(0, 1, size(t,2));
13 A = spcol(knots, k, t1);
14 coefs1 = A\P;
15 plot(coefs1(:,1), coefs1(:,2), 'k--o');
16 x = spmak(knots, coefs1.');
17 fnplt(x);
18 function [t] = uniform_param(a, b, n)

```

3 Superfici

3.1 Superfici parametriche

Una superficie può essere rappresentata attraverso la forma parametrica:

$$\mathbf{X}(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}$$

con $u, v \in [a, b] \subset \mathbb{R}^2$. La seguente superficie parametrica

$$\mathbf{X}(u, v) = \begin{pmatrix} x(u, v) = (2 + \cos(v)) \cdot \cos(u) \\ y(u, v) = (2 + \cos(v)) \cdot \sin(u) \\ z(u, v) = \sin(v) \end{pmatrix}$$

è stata implementata in Codice 9. Il plot è visualizzato in Figura 17.

Codice 9: Disegno di una superficie parametrica

```

1 u = linspace(0, 10, 100);
2 v = linspace(0, 10, 100);
3 [uu, vv] = meshgrid(u, v);
4 x = (2 + cos(vv)).*cos(uu);
5 y = (2 + cos(vv)).*sin(uu);
6 z = sin(vv);
7 surf(x, y, z); axis equal;

```

3.2 Superfici *Tensor-Product* di Bézier

Definizione 2. *Dati due spazi di funzioni monovariate:*

$$S_1 = \langle B_0^A(u), \dots, B_n^A(u) \rangle \quad S_2 = \langle B_0^B(v), \dots, B_m^B(v) \rangle$$

con $u \in [a_A, b_A]$ e $v \in [a_B, b_B]$ si definisce uno spazio *Tensor-Product* di funzioni bivariate come:

$$S_1 \times S_2 = \left\{ f : \mathbb{R} \rightarrow \mathbb{R} : f(u, v) = \sum_{i=0}^n \sum_{j=0}^m c_{i,j} B_i^A(u) B_j^B(v) \right\}$$

con $c_{i,j} \in \mathbb{R}$.

Una superficie *Tensor-Product* di Bézier si definisce a partire dalla base di Bézier e da $(n+1) \cdot (m+1)$ vertici di controllo, i quali a loro volta formano un poligono di controllo.

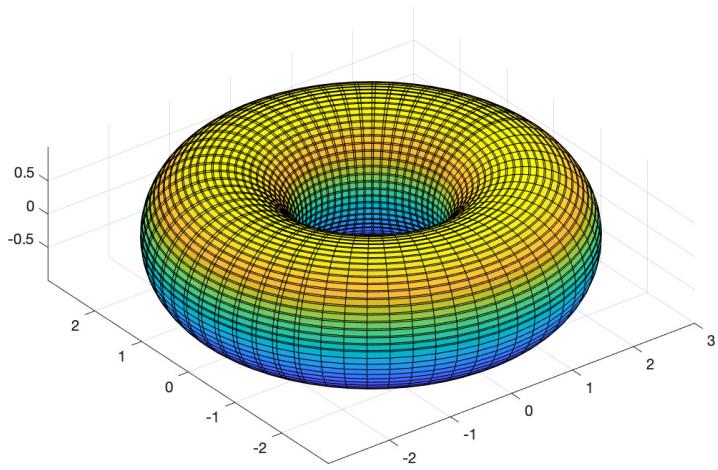


Figura 17: Superficie parametrica

Definizione 3. Una superficie Tensor-Product di Bézier è data da:

$$\mathbf{X}(u, v) = \sum_{i=0}^n \sum_{j=0}^m \mathbf{b}_{i,j} B_i^n(u) B_j^m(v)$$

dove:

- B_i^n e B_j^m sono i polinomi di Bernstein rispettivamente di grado n e m e indice i e j .
- $\mathbf{b}_{i,j}$ sono gli $(n+1) \cdot (m+1)$ vertici di controllo.

Di seguito, in Codice 10 un'implementazione della base per le superfici di Bézier realizzata con l'uso della funzione `spcol`.

Codice 10: Base delle superfici di Bézier

```

1 k_u = 3; k_v = 3;
2 knots_u = [zeros(1, k_u), ones(1, k_u)];
3 knots_v = [zeros(1, k_v), ones(1, k_v)];
4 tab = 0:0.05:1;
5 B_u = spcol(knots_u, k_u, tab);
6 B_v = spcol(knots_v, k_v, tab);
7 for i = 1:length(knots_u)-k_u
8     for j = 1: length(knots_v)-k_v

```

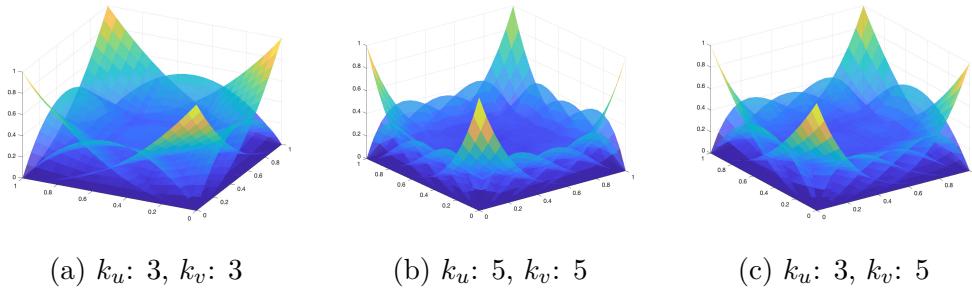


Figura 18: Base di Bézier

```

9      X = B_u(:,i)*B_v(:,j).';
10     surf(tab, tab, X, 'FaceAlpha', 0.8); shading flat;
11     s.EdgeColor = 'none';
12     hold on;
13 end
14 end

```

Codice 11: Disegno di una superficie di Bézier

```

1 k_u = 3; k_v = 3;
2 b_x = [1 2 3; 1 2 4; 1 2 4];
3 b_y = [4 6 4; 3 5 2; 3 2 1];
4 b_z = [2 2 2; 2 4 2; 2 5 3];
5 plot_control_pol(b_x, b_y, b_z); grid on; axis tight; axis
    equal;
6 knots_u = [zeros(1, k_u), ones(1, k_u)];
7 knots_v = [zeros(1, k_v), ones(1, k_v)];
8 tab = 0:0.05:1;
9 B_u = spcol(knots_u, k_u, tab);
10 B_v = spcol(knots_v, k_v, tab);
11 X_x = B_u*b_x*B_v.';
12 X_y = B_u*b_y*B_v.';
13 X_z = B_u*b_z*B_v.';
14 surf(X_x, X_y, X_z, 'FaceAlpha', 0.8); shading flat; s.
    EdgeColor = 'none';
15 hold on;
16 plot3(X_x(1, 1), X_y(1,1), X_z(1,1), 'k.', 'MarkerSize', 20);
17 plot3(X_x(end, end), X_y(end, end), X_z(end, end), 'k.', '
    MarkerSize', 20);
18 plot3(X_x(end, 1), X_y(end,1), X_z(end,1), 'k.', 'MarkerSize',
    20);

```

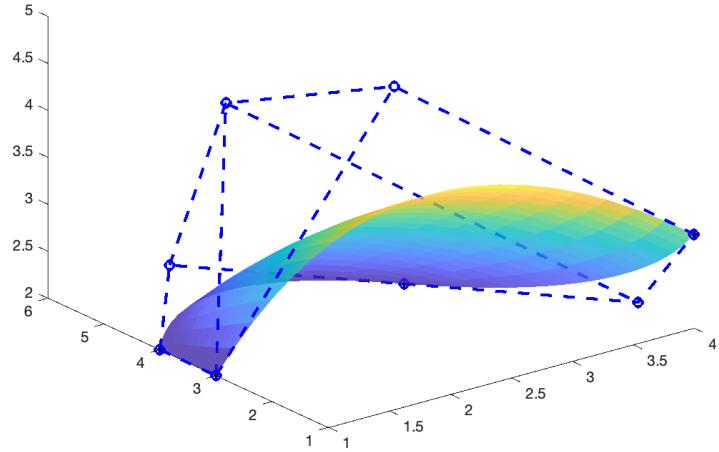


Figura 19: Superficie di Bézier generata da Codice 11

3.2.1 Proprietà delle superfici di Bézier

Come per le curve, le superfici di Bézier godono di alcune proprietà:

- Invarianza per trasformazioni affini: applicare una trasformazione affine sui punti di controllo o sulla superficie è indifferente, il risultato sarà uguale.
- Curve di bordo: le quattro curve di bordo della superficie di Bézier sono date da

$$\mathbf{X}(0, v) = \sum_{j=0}^m \mathbf{b}_{0,j} B_j^m(v) \quad \mathbf{X}(1, v) = \sum_{j=0}^m \mathbf{b}_{m,j} B_j^m(v)$$

$$\mathbf{X}(u, 0) = \sum_{i=0}^n \mathbf{b}_{i,0} B_i^n(u) \quad \mathbf{X}(u, 1) = \sum_{i=0}^n \mathbf{b}_{i,n} B_i^n(u)$$

Invarianza per trasformazioni affini Come detto in precedenza questa proprietà dice che applicare una trasformazione affine sui punti di controllo o sulla superficie è indifferente. Questa proprietà risulta molto utile nella situazione in cui si deve applicare una determinata trasformazione ad una superficie. Il modo più efficiente per realizzare ciò è applicare la trasformazione ai punti di controllo e successivamente ridisegnare la superficie. In Codice 12 è possibile vedere l'esempio di un disegno di una superficie e successiva trasformazione, sia sui punti di controllo che sulla superficie stessa.

Codice 12: Trasformazione affine

```

1 k_u = 3; k_v = 3;
2 b_x = [1 2 3; 1 2 4; 1 2 4];
3 b_y = [4 6 4; 3 5 2; 3 2 1];
4 b_z = [2 2 2; 2 4 2; 2 5 3];
5 plot_control_pol(b_x, b_y, b_z); grid on; axis tight; axis
    equal;
6 knots_u = [zeros(1, k_u), ones(1, k_u)];
7 knots_v = [zeros(1, k_v), ones(1, k_v)];
8 tab = 0:0.05:1;
9 B_u = spcol(knots_u, k_u, tab);
10 B_v = spcol(knots_v, k_v, tab);
11 X_x = B_u*b_x*B_v.';
12 X_y = B_u*b_y*B_v.';
13 X_z = B_u*b_z*B_v.';
14 surf(X_x, X_y, X_z, 'FaceAlpha', 0.8); shading flat; s.
    EdgeColor = 'none';
15 hold on;
16 plot3(X_x(1, 1), X_y(1,1), X_z(1,1), 'k.', 'MarkerSize', 20);
17 plot3(X_x(end, end), X_y(end, end), X_z(end, end), 'k.', '
    MarkerSize', 20);
18 plot3(X_x(end, 1), X_y(end,1), X_z(end,1), 'k.', 'MarkerSize',
    20);
19 plot3(X_x(1, end), X_y(1,end), X_z(1,end), 'k.', 'MarkerSize',
    20);
20 %trasformazione sui punti
21 theta = pi/2;
22 A = [cos(theta) -sin(theta) 0 ; sin(theta) cos(theta) 0; 0 0
    1];
23 new_points = [b_x(:)'; b_y(:)'; b_z(:)'];
24 for i = 1:size(new_points,2)
25     new_points(:, i) = A*new_points(:,i);
26 end
27 X_x2 = B_u*(reshape(new_points(1, :), k_u, k_v))*B_v.';
28 X_y2 = B_u*(reshape(new_points(2, :), k_u, k_v))*B_v.';
29 X_z2 = B_u*(reshape(new_points(3, :), k_u, k_v))*B_v.';
30 surf(X_x2, X_y2, X_z2, 'FaceAlpha', 0.8);
31 plot_control_pol(reshape(new_points(1, :), k_u, k_v), ...
    reshape(new_points(2, :), k_u, k_v), ...
    reshape(new_points(3, :), k_u, k_v));
32 %trasformazione sulla superficie
33 new_sup = [X_x(:)'; X_y(:)'; X_z(:)'];
34 for i = 1:size(new_sup,2)
35     new_sup(:, i) = A*new_sup(:,i);
36 end
37 X_x3 = B_u*(reshape(new_points(1, :), k_u, k_v))*B_v.';
38 X_y3 = B_u*(reshape(new_points(2, :), k_u, k_v))*B_v.';
39 X_z3 = B_u*(reshape(new_points(3, :), k_u, k_v))*B_v.';
```

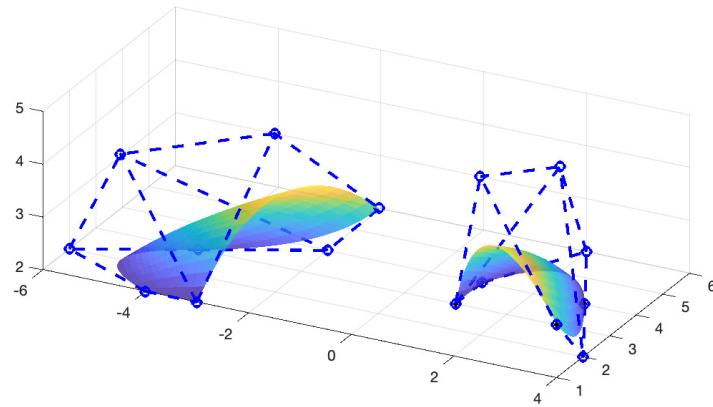


Figura 20: Trasformazione affine su una superficie di Bézier

```

42 surf(X_x3, X_y3, X_z3, 'FaceAlpha', 0.8); shading flat;
43 isequal(X_x2, X_x3)
44 isequal(X_y2, X_y3)
45 isequal(X_z2, X_z3)

```

In questo caso per mostrare che le due superfici trasformate sono uguali è stata usata la funzione **isequal**, che restituisce 1 se e solo se le due matrici in input sono uguali. Il plot in output del Codice 12 è mostrato in Figura 20.

Curve di bordo I bordi di una superficie di Bézier possono essere visti a loro volta come quattro curve di Bézier:

$$\begin{aligned}\mathbf{X}(0, v) &= \sum_{j=0}^m \mathbf{b}_{0,j} B_j^m(v) & \mathbf{X}(1, v) &= \sum_{j=0}^m \mathbf{b}_{m,j} B_j^m(v) \\ \mathbf{X}(u, 0) &= \sum_{i=0}^n \mathbf{b}_{i,0} B_i^n(u) & \mathbf{X}(u, 1) &= \sum_{i=0}^n \mathbf{b}_{i,m} B_i^n(u)\end{aligned}$$

In Codice 13 è possibile vedere il calcolo, tramite l'algoritmo di *De Casteljau*, di una delle quattro curve di bordo, mentre in Figura 21 è possibile vedere il plot dei bordi di una superficie di Bézier.

Codice 13: Curve di bordo

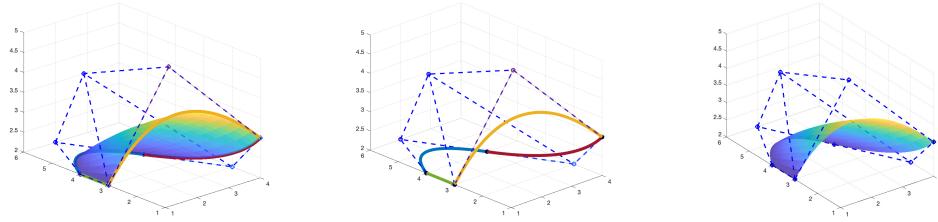


Figura 21: Curve di bordo

```

1 x_p = b_x(:,1); y_p = b_y(:,1); z_p = b_z(:,1);
2 for i = 1:length(u)
3     [t_x, t_y, t_z] = de_casteljau(k_u, x_p, y_p, z_p, u(i));
4     p_x(i) = t_x(k_u, k_u);
5     p_y(i) = t_y(k_u, k_u);
6     p_z(i) = t_z(k_u, k_u);
7 end
8 plot3(x_p, y_p, z_p, '-o'); hold on;
9 plot3(p_x, p_y, p_z, 'linewidth', 5); hold on;

```

3.2.2 Algoritmo di De Casteljau

Una superficie di Bézier, come definita in Definizione 3, è possibile ottenerla tramite l'algoritmo di *De Casteljau* eseguendo la seguente interpolazione bilineare.

$$\mathbf{b}_{i,j}^0 = b_{i,j} \quad i = 0, \dots, n \quad j = 0, \dots, m$$

$$\mathbf{b}_{i,j}^{r,s}(u, v) = ((1-u) \quad u) \begin{pmatrix} \mathbf{b}_{i,j}^{r-1,s-1} & \mathbf{b}_{i,j+1}^{r-1,s-1} \\ \mathbf{b}_{i+1,j}^{r-1,s-1} & \mathbf{b}_{i+1,j+1}^{r-1,s-1} \end{pmatrix} \begin{pmatrix} (1-v) \\ v \end{pmatrix}$$

Eseguendo questa operazione sopra descritta si procede contemporaneamente in entrambe le direzioni r ed s , quindi può essere eseguita un numero di volte massimo pari a $\gamma = \min(n, m)$, dopodiché è necessario procedere in una sola direzione eseguendo le iterazioni rimanenti. L'approccio scelto per implementare l'algoritmo di *De Casteljau* è descritto in The NURBS Book. L'algoritmo, implementato in Codice 14, *non* procede in entrambe le direzioni contemporaneamente, bensì prima in una e poi nell'altra. Acciocché questo possa essere reso possibile, è necessario dapprima implementare l'algoritmo per il disegno di curve per poi, successivamente, sfruttarlo per disegnare la superficie.

Codice 14: Algoritmo di De Casteljau

```

1 P = zeros(n,m,3);

```

```

2 P(:,:,1) = [1 3 3; 1 3 3; 1 3 3];
3 P(:,:,2) = [1 1 1; 4 4 4; 3 3 3];
4 P(:,:,3) = [2 1 2; 2 2 2; 0 1 1];
5 plot3(P(:,:,1), P(:,:,2), P(:,:,3), 'k--o'); hold on;
6 plot3(P(:,:,1)', P(:,:,2)', P(:,:,3)', 'k--o')
7 knots_u = [zeros(1, n), ones(1, n)];
8 knots_v = [zeros(1, m), ones(1, m)];
9 tab = 0:0.05:1;
10 B_u = spcol(knots_u, n, tab); B_v = spcol(knots_v, m, tab);
11 surface(:,:,1) = B_u*P(:,:,1)*B_v.';
12 surface(:,:,2) = B_u*P(:,:,2)*B_v.';
13 surface(:,:,3) = B_u*P(:,:,3)*B_v.';
14 surf(surface(:,:,1), surface(:,:,2), surface(:,:,3), 'FaceAlpha
', 0.8);
15 for u0 = 0:0.05:1
16     for v0 = 0:0.05:1
17         s = deCasteljau2(P, n, m, u0, v0);
18         plot3(s(1), s(2), s(3), 'r.', 'markersize', 15)
19     end
20 end
21 function [s] = deCasteljau2(P, n, m, u0, v0)
22     Q = zeros(1, max(n,m), 3);
23     if(n>m)
24         for j = 1:m
25             Q(:,j,:) = deCasteljau1(permute(P(:,j,:),[2 1 3]),
n, u0);
26         end
27         s = deCasteljau1(Q, m, v0);
28     else
29         for i = 1:n
30             Q(:,i,:) = deCasteljau1(P(i,:,:), m, v0);
31         end
32         s = deCasteljau1(Q, n, u0);
33     end
34 end
35 function [c] = deCasteljau1(P, n, u)
36     Q = P;
37     for k = 2:n+1
38         for i = 1:n-k+1
39             Q(:,i,:) = (1-u)*Q(:,i,:)+u*Q(:,i+1,:);
40         end
41     end
42     c = [Q(1,1,1), Q(1,1,2), Q(1,1,3)]';
43 end

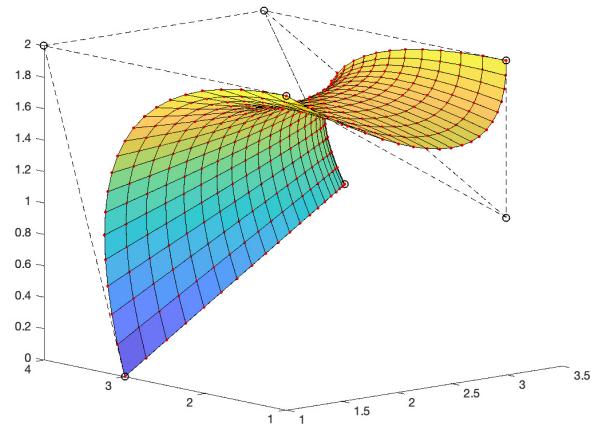
```

Per accettarsi del corretto funzionamento dell'algoritmo implementato in Codice 14 sono state provate tutte le possibili casistiche riguardanti i gradi, ovvero:

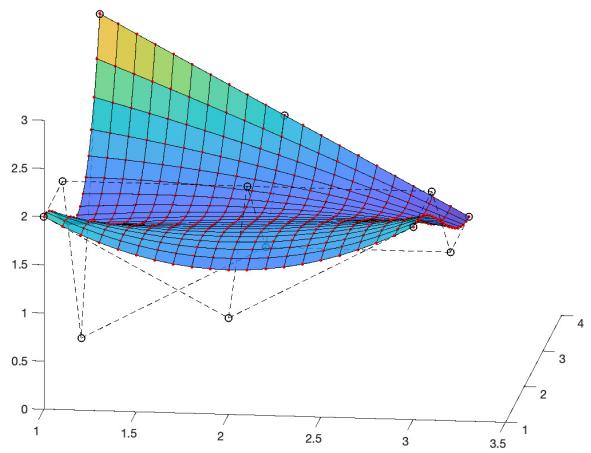
- $m > n$

- $n > m$
- $m = n$

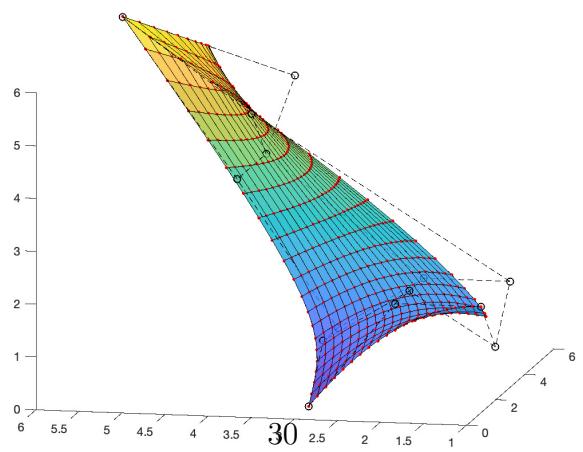
In Figura 22 è possibile vedere le tre diverse casistiche per i gradi delle superfici. I *pallini* in rosso sono i punti calcolati con l'algoritmo di *De Casteljau*, e come **speciale** sia possibile evidenziare si sovrappongono alla superficie disegnata in maniera convenzionale.



(a) $m = 3, n = 3$



(b) $m = 3, n = 4$



(c) $m = 5, n = 3$

Figura 22: Superfici disegnate con *De Casteljau*