



UNIVERSITÀ
DEGLI STUDI
FIRENZE

SCIENZE MATEMATICHE FISICHE E NATURALI:
LAUREA MAGISTRALE IN INFORMATICA

CORSO DI METODI NUMERICI PER LA GRAFICA

Esercizi in MatLab

Autore:
Andrea MOSCATELLI

Docenti:
Carlotta GIANNELLI
Costanza CONTI

Anno accademico 2017-2018

Contents

1 Curve B-Spline in 2D	2
1.1 Basi B-Spline	2
1.2 Curve B-Spline	5
1.3 Algoritmo di De Boor	8
1.4 Algoritmo di knot insertion	10
2 Superfici parameteriche	14
2.1 Area della superficie	14
2.2 Superfici tensor product - Bezier	15
2.3 Algoritmo di De Casteljau su superfici tensor product Bezier . . .	17
2.4 Superfici tensor product - B-Spline	20
2.5 Algoritmo di De Boor su superfici tensor product B-Spline	23
3 Interpolazione	27
3.1 Interpolazione 2D	27
3.2 Interpolazione 3D	28
4 Hierarchical B-Spline	31
4.1 Basi HB-Spline	31

1 Curve B-Spline in 2D

1.1 Basi B-Spline

Disegnare le basi B-Spline di ordine k (a scelta) per i seguenti vettori di nodi:

- (1, 1.2, 3, 4.5, 4.8, 6) clumped
- (1, 2, 3, 4, 5) uniforme
- (1, 2, 3, 4, 5) clumped

Per ognuno di essi mostrare che le basi ottenute sono una partizione dell'unità.

Svolgimento

Dato un vettore esteso dei nodi $\mathbf{t} = [t_1, t_2, \dots, t_{n+k}]$, le basi B-Spline di ordine 1 sono definite come segue:

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

mentre le basi di ordine $r \leq k$ sono definite ricorsivamente come segue:

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

dove

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

L'implementazione matlab dell'esercizio richiesto e di quanto definito è la seguente (output del codice in figura 1, 2, 3):

```
1 ab = [0,6];
2 knots = [1,2,3,4,5];
3 m = [1,1,1,1,1];
4 k = 2; % ordine
5
6 #####%
7
8 subplot(3,1,1);
9 extendedKnots = augknt([0,1,1.2,3,4.5,4.8,6],k);
10 plotBases(extendedKnots,k)
11 title('bases for specified nodes')
12
13
14 subplot(3,1,2);
15 extendedKnots = getClumpedExtendedPartition(ab,knots,k,m)
16 plotBases(extendedKnots,k)
17 title('bases for clumped nodes')
```

```

18
19 subplot(3,1,3);
20 extendedKnots = getUniformExtendedPartition(ab,knots,k,m)
21
22 plotBases(extendedKnots,k)
23 title('bases for uniform nodes')
24
25
26
27 function plotBases(extendedKnots,k)
28 M = length(extendedKnots)-k*2;
29 spaceDimension = k+M;
30 allBases = 0;
31 points = 500;
32 column = 1;
33 for t = linspace(extendedKnots(1),extendedKnots(end),points)
34     for i = 1:spaceDimension
35         allBases(i, column) = BSpline(i,k,t,extendedKnots);
36     end
37     column = column + 1;
38 end
39
40 hold off;
41 total = sum(allBases(:,1:points))
42 plot(linspace(extendedKnots(1),extendedKnots(end),points),total,'k','
43     ↪ LineWidth',3);
44 hold on
45 xValues = linspace(extendedKnots(1),extendedKnots(end),points);
46 for i = 1:spaceDimension
47     plot(xValues, allBases(i,:), 'color',rand(1,3), 'LineWidth',2);
48     hold on;
49 end
50 hold off;
51 end
52
53 function bspline = BSpline(i, r, t, tVect)
54 if r==1
55     if (t >= tVect(i) && t < tVect(i+1) || t == tVect(i+1) && tVect(i+1)
56         ↪ == tVect(end))
57         bspline = 1;
58     else
59         bspline = 0;
60     end
61 else
62     t1 = (t-tVect(i))/(tVect(i+r-1) - tVect(i));
63     if(isnan(t1) || isinf(t1)) t1 = 0;
64     end
65     t2 = (tVect(i+r) - t)/(tVect(i+r) - tVect(i+1));
66     if(isnan(t2)|| isinf(t2)) t2 = 0;
67     end

```

```

66    bsppline = t1 * BSpline(i, r-1, t, tVect) + t2 * BSpline(i+1, r-1, t,
67        ↪ tVect);
68 end
end

```

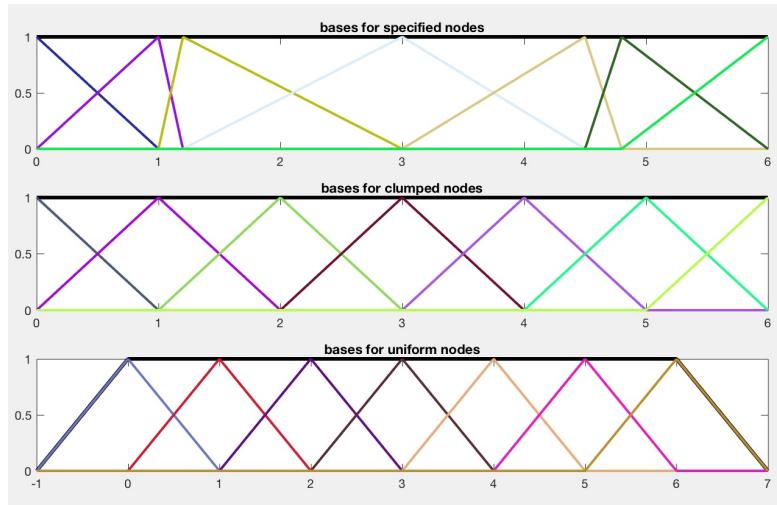


Figure 1: Basi B-Spline di ordine 2 per i vettori di nodi specificati e loro somma (in nero)

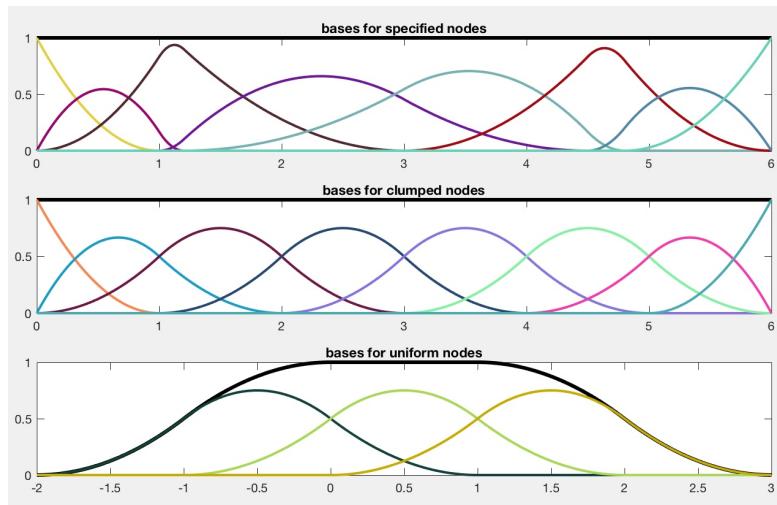


Figure 2: Basi B-Spline di ordine 3 per i vettori di nodi specificati e loro somma (in nero)

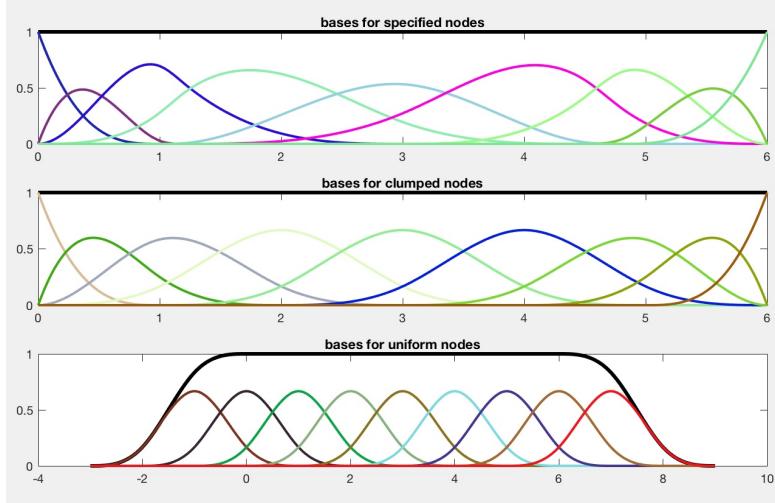


Figure 3: Basi B-Spline di ordine 4 per i vettori di nodi specificati e loro somma (in nero)

1.2 Curve B-Spline

Scelti un ordine k , un vettore di nodi e dei coefficienti bidimensionali a piacere, disegnare:

- la relativa curva B-Spline utilizzando la funzione implementata nell'esercizio precedente
- la relativa curva B-Spline utilizzando il *Curve Fitting Toolbox* di Matlab
- la relativa curva B-Spline chiusa.

Svolgimento

Dati un vettore esteso di nodi $\mathbf{t} = [t_1, t_2, \dots, t_{n+k}]$ e un insieme di vertici di controllo (coefficients) $\mathbf{d} = [d_1, d_2, \dots, d_n]$ la relativa curva B-Spline è data dalla formula parametrica

$$X(t) = \sum_{i=0}^n d_i N_{i,k}(t)$$

Se vogliamo invece rappresentare una curva chiusa con gli stessi coefficienti, dobbiamo ripetere $k-1$ punti di controllo, ovvero

$$d_{n+2-k-i} = d_i \quad i = 0, \dots, k-2$$

e mantenere l'ampiezza degli intervalli $h_i = [t_i, t_{i+1}]$, secondo la regola

$$h_i = h_{n+2-k+i}, \quad h_{n+1+i} = h_{k-1+i} \quad i = 0, \dots, k-2$$

L'implementazione Matlab dell'esercizio richiesto qui di seguito e output in figura 4, 5 e 6.

```

1 k = 3; % ordine
2 ab = [1,2,3,4];
3
4 #####%
5 extendedKnots = augknt(ab,k);
6
7 M = length(extendedKnots)-((k-1)*2)-2; % sum(mVector)
8 coefs = ginput(M+k);
9
10 points = getBsplineCurve(ab,k,coefs)
11
12 plot(coefs(:,1),coefs(:,2),'b',coefs(:,1),coefs(:,2),'bo')
13 hold on
14 plot(points(:,1),points(:,2),'r','LineWidth',3)
15 title('BSpline curve with algorithm')
16
17 ginput(1);
18 clf;
19 plot(coefs(:,1),coefs(:,2),'b',coefs(:,1),coefs(:,2),'bo')
20 hold on
21 drawBspline(k, extendedKnots,coefs)
22 title('BSpline curve with curve fitting toolbox')
23
24 ginput(1);
25 clf;
26 extendedKnots = [ab(1)-k+1:ab(1), ab(2:end-1),ab(end):ab(end)+2*k-2];
27 coefs = [coefs;coefs(1:k-1,:)]
28 drawBspline(k, extendedKnots,coefs)
29 hold on
30 plot(coefs(:,1),coefs(:,2),'b',coefs(:,1),coefs(:,2),'bo')
31 title('Closed Bspline with curve fitting toolbox')
32
33 function points = getBsplineCurve(ab,k,coefs)
34 extendedKnots = augknt(ab,k)
35 idx = 1;
36 nc = length(coefs);
37 for t = linspace(ab(1),ab(end),100)
38 point = 0;
39 for i = 1:nc
40 Vi = coefs(i,:);
41 Nik = BSpline(i,k,t,extendedKnots);
42 point = point + (Vi * Nik);
43 end
44 points(idx,:) = point;
45 idx = idx + 1;
46 end

```

```

48 end
49
50 function [extendedKnots, coefs] = drawBspline(order, extendedKnots,coefs)
51 coefs = coefs';
52 x = spmak(extendedKnots,coefs);
53 fnplt(x,[extendedKnots(order),extendedKnots(end-(order-1))], 'k');
54 end

```

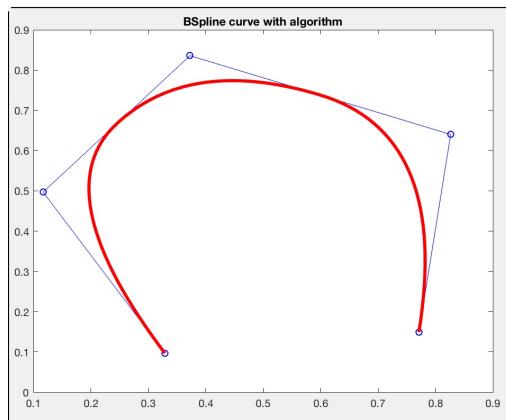


Figure 4: Curva B-Spline disegnata col metodo implementato nell'esercizio precedente

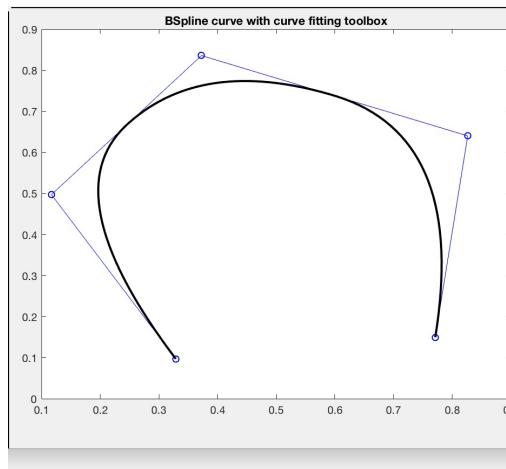


Figure 5: Curva B-Spline disegnata utilizzando gli stessi coefficienti e ordine della figura precedente e il Curve Fitting Toolbox di Matlab

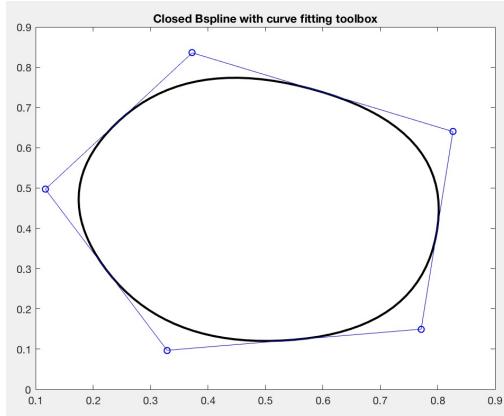


Figure 6: Curva B-Spline chiusa ottenuta utilizzando gli stessi coefficienti e ordine delle figure precedenti

1.3 Algoritmo di De Boor

Scelti un ordine k , un vettore di nodi, dei coefficienti bidimensionali a piacere ed un punto $t \in [0,1]$, disegnare:

- il punto della relativa curva B-Spline associato a quel valore t calcolato utilizzando l'algoritmo di de Boor
- la relativa curva B-Spline ottenuta calcolando ogni suo punto con l'algoritmo di de Boor e verificare che sia equivalente a quella ottenuta col *Curve Fitting Toolbox* di Matlab.

Svolgimento

Data la proprietà di località di una curva B-Spline, si osserva che per valutare il punto $X(t)$ con $t \in [t_j, t_{j+1}]$, basterà calcolare:

$$X(t) = \sum_{i=j-k+1}^j d_i N_{i,k}(t)$$

L'algoritmo di de Boor si basa sul fatto che è possibile riscrivere questa formula utilizzando basi B-Spline diminuite di un grado, ovvero:

$$X(t) = \sum_{i=j-k+2}^j d_i^{(2)} N_{i,k-1}(t)$$

dove

$$d_i^{(2)}(t) = (1 - \omega_{i,k-1}(t)) * d_{i-1}(t) + \omega_{i,k-1}(t) * d_i(t).$$

Iterando il procedimento r volte, con $r = 1, \dots, (k - 1)$, al passo r-esimo avremo:

$$X(t) = \sum_{i=j-k+1+r}^j d_i^{(r)} N_{i,k-r}(t)$$

dove

$$d_i^{(r)}(t) = (1 - \omega_{i,k-r+1}(t)) * d_{i-1}^{(r-1)}(t) + \omega_{i,k-r+1}(t) * d_i^{(r-1)}(t).$$

All'ultimo passo, ovvero al (k-1)-esimo, otteniamo:

$$X(t) = \underbrace{\sum_{i=j-k+(k-1)+1}^j d_i^{(k-1)}}_{i=j} \underbrace{N_{i,1}(t)}_1 = d_j^{(k-1)}(t)$$

L'implementazione Matlab di quanto richiesto è la seguente (output in figura 7):

```

1 knots = [1,2,3];
2 k = 5;
3 t = 1.5;
4
5 #####%
6 clf;
7 extendedKnots = augknt(knots,k)
8
9 M = length(knots)-2; % sum(mVector)
10 coefs = ginput(M+k);
11 vX = coefs(:,1);
12 vY = coefs(:,2);
13
14 plot(vX,vY, 'b', vX,vY, 'bo')
15
16 [x,y] = getPointDeBoor(t, extendedKnots, k, vX, vY);
17 hold on
18 plot(x,y, 'ro');
19
20 xValues = 0;
21 yValues = 0;
22 for j = knots(1):0.001:knots(end)
23     [x,y] = getPointDeBoor(j,extendedKnots,k,vX,vY);
24     xValues(end + 1) = x;
25     yValues(end + 1) = y;
26 end
27
28 plot(xValues(2:end),yValues(2:end), 'g', 'LineWidth',3);
29
% verifica che sia la stessa curva disegnata usando Curve fitting toolbox

```

```

31 hold on
32 drawBspline(k,extendedKnots,coefs)
33
34 title('Point calculated with De Boor algorithm')
35
36 function [xValue,yValue] = getPointDeBoor(t, extendedKnots, k, vX, vY)
37 j = find(extendedKnots <= t);
38 j = j(end);
39 j = min(j,length(extendedKnots)-k+1);
40 xValue = getVectorDerivate(vX, j, k-1, k, t, extendedKnots);
41 yValue = getVectorDerivate(vY, j, k-1, k, t, extendedKnots);
42 end
43
44
45 function derivative = getVectorDerivate(vertexVector, i, r, k, t,
46     ↪ extendedKnots)
46 if r == 0
47     if i <= length(vertexVector) && i > 0
48         derivative = vertexVector(i);
49     else
50         derivative = 0;
51     end
52 else
53     drm1im1 = getVectorDerivate(vertexVector,i-1, r-1, k, t, extendedKnots);
54     drm1i = getVectorDerivate(vertexVector,i, r-1, k, t, extendedKnots);
55     t1 = (t-extendedKnots(i));
56     if (extendedKnots(i+k-r) ~= extendedKnots(i))
57         t1 = t1 / (extendedKnots(i+k-r) - extendedKnots(i));
58     end
59     t2 = 1-t1;
60     derivative = t1 * drm1i + t2 * drm1im1;
61 end
62 end

```

1.4 Algoritmo di knot insertion

Scelti un ordine k , un vettore dei nodi e dei coefficienti bidimensionali a piacere applicare l'algoritmo di *knot insertion* e verificare che la curva generata dal nuovo poligono di controllo sia identica a quella generata dal poligono di controllo originario.

Svolgimento

Se vogliamo inserire un nuovo vertice di controllo al nostro poligono dobbiamo come prima cosa inserire un nuovo valore \hat{t} al nostro vettore dei nodi \mathbf{t} che diventerà quindi:

$$\hat{\mathbf{t}} = [t_1, t_2, \dots, t_j, \hat{t}, t_{j+1}, \dots, t_{n+k}].$$

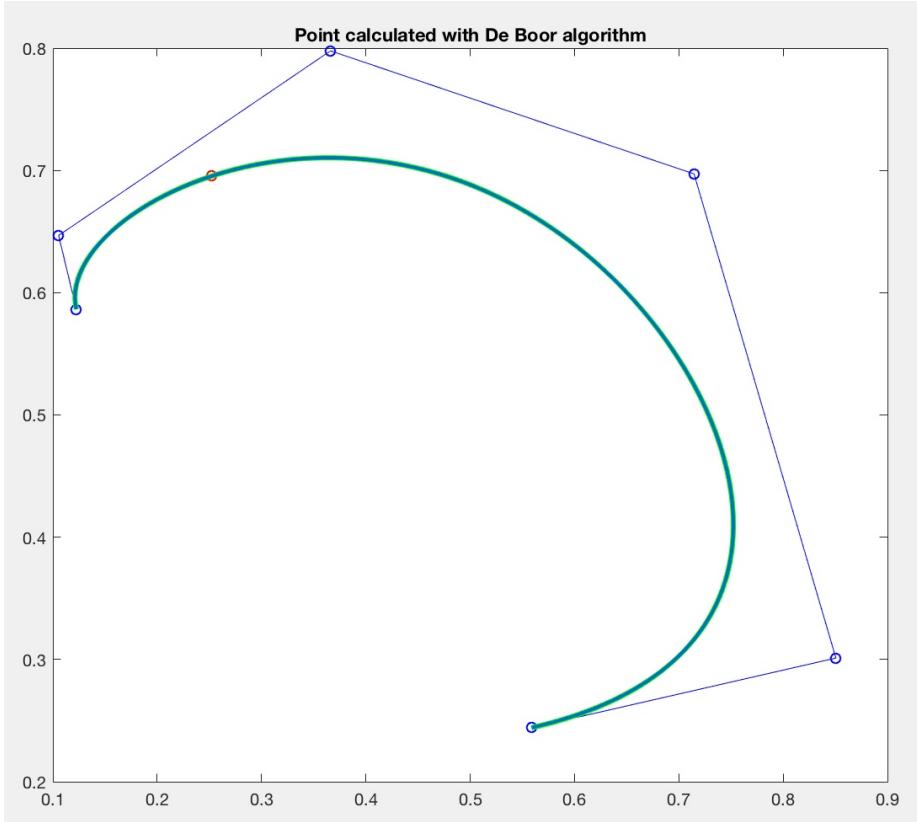


Figure 7: Punto della curva calcolato con l'algoritmo di de Boor (cerchiato in rosso); intera curva calcolata con algoritmo di de Boor (linea verde); curva calcolata con Curve Fitting Toolbox di matlab (linea nera, che combacia con quella precedente)

Il problema si riduce quindi nel trovare quegli $(n+1)$ coefficienti $\hat{\mathbf{d}}$ per i quali

$$X(t) = \sum_{i=j-k+2}^j \hat{d}_i \hat{N}_{i,k-1}(t)$$

dove le $\hat{N}_{i,k-1}$, $i = 0, \dots, (n+1)$ formano la base dello spazio B-Spline col vettore esteso dei nodi \hat{t} .

Per la proprietà della località ci accorgiamo subito che $\hat{d}_i = d_i$ se $i = 0, \dots, (j - k)$ e $\hat{d}_i = d_{i-1}$ se $i = j + 1, \dots, (n + 1)$ e che i $(k+1)$ punti rimanenti non sono altro che una combinazione lineare del punto di controllo precedente

con quello successivo, ovvero:

$$\hat{d} = \begin{cases} d_i & \text{se } i = 0, \dots, j-k \\ (1 - \omega_{1,k}(\hat{t})) * d_{i-1} + \omega_{1,k}(\hat{t}) * d_i & \text{se } i = j-k+2, \dots, j \\ d_{i-1} & \text{se } i = j+1, \dots, n+1 \end{cases}$$

L'implementazione di quanto richiesto è la seguente (output in figura 8):

```

1 k = 5;
2 t = 2.7;
3
4 % #####
5 clf;
6
7 extendedKnots = augknt(knots,k)
8
9 M = length(knots)-2; % sum(mVector)
10 coefs = ginput(M+k)
11 vX = coefs(:,1);
12 vY = coefs(:,2);
13
14 plot(vX,vY,'b',vX,vY,'bo', 'LineWidth',6)
15 hold on
16 drawBspline(k,extendedKnots,coefs,6)
17
18 [vXnew, vYnew, extendedKnotsNew] = insertKnot(t, extendedKnots, vX,vY, k
19   ↪ );
20
21 hold on
22 plot(vXnew,vYnew,'r',vXnew,vYnew,'ro', 'LineWidth',2)
23 hold on
24 drawBspline(k,extendedKnotsNew,[vXnew,vYnew]',2)
25
26 function [vXnew, vYnew, extendedKnotsNew] = insertKnot(t, extendedKnots,
27   ↪ vX,vY, k)
28 l = find(extendedKnots <= t);
29 l = l(end);
30 l = min(l,length(extendedKnots)-k+1);
31 extendedKnotsNew = [extendedKnots(1:l) t extendedKnots((l+1):end)];
32
33 for i = 1: (length(vX)+1)
34   if i <= (l-k+1)
35     vXnew(i) = vX(i);
36     vYnew(i) = vY(i);
37   else if i <= l
38     wikt = getW(i, k, t, extendedKnots);
39     vXnew(i) = (1-wikt) * vX(i-1) + wikt * vX(i);
40     vYnew(i) = (1-wikt) * vY(i-1) + wikt * vY(i);
41   else
42
43 end
44
45 % #####
46
```

```

41         vXnew(i) = vX(i-1);
42         vYnew(i) = vY(i-1);
43     end
44 end
45 end
46
47 function wirt = getW(i, r, t, tVect)
48 if t >= tVect(i+r-1)
49     wirt = 0;
50 else
51     wirt = (t - tVect(i));
52     if(tVect(i+r-1) ~= tVect(i))
53         wirt = wirt / (tVect(i+r-1) - tVect(i));
54     end
55 end
56 end
57 end

```

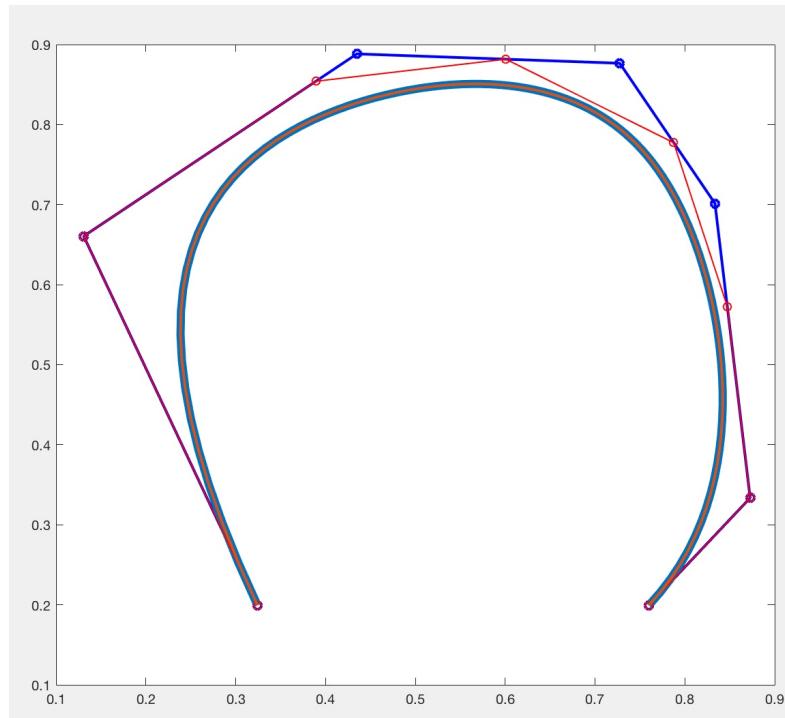


Figure 8: In blu il poligono di controllo originale e la relativa curva B-Spline. In rosso il poligono aumentato di un nodo con la relativa curva B-Spline (sovraposta alla precedente)

2 Superfici parameteriche

2.1 Area della superficie

Definire la formula parametrica $\mathbf{X}(u,v)$ della sfera (o del paraboloide) e calcolare l'area della superficie nell'intervallo $\bar{u} = [3, 6]$, $\bar{v} = [-0.5, 1]$.

Svolgimento

Data la formula parametrica della sfera

$$\mathbf{X}(u, v) = \begin{pmatrix} \cos(u)\cos(v) \\ \sin(u)\cos(v) \\ \sin(v) \end{pmatrix} \quad u \in [0, 2\pi] \quad v \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

e considerando la formula dell'area di una superficie

$$A = \int \int |\mathbf{X}_u \wedge \mathbf{X}_v| dudv = \int \int \sqrt{EG - F^2} dudv$$

dove

$$\mathbf{X}_u = \begin{pmatrix} \frac{\partial x(u,v)}{\partial u} \\ \frac{\partial y(u,v)}{\partial u} \\ \frac{\partial z(u,v)}{\partial u} \end{pmatrix}; \quad \mathbf{X}_v = \begin{pmatrix} \frac{\partial x(u,v)}{\partial v} \\ \frac{\partial y(u,v)}{\partial v} \\ \frac{\partial z(u,v)}{\partial v} \end{pmatrix}; \quad \begin{aligned} E &= E(u, v) = \mathbf{X}_u \cdot \mathbf{X}_u \\ F &= F(u, v) = \mathbf{X}_u \cdot \mathbf{X}_v \\ G &= G(u, v) = \mathbf{X}_v \cdot \mathbf{X}_v \end{aligned}$$

Possiamo implementare l'esercizio come segue (output in figura 9):

```

1 syms u v;
2 % sfera
3 X = cos(u) .* cos(v);
4 Y = sin(u) .* cos(v);
5 Z = sin(v);
6
7 % paraboloide
8 % X = v * cos(u);
9 % Y = v * sin(u);
10 % Z = v^2;
11
12 % range parametri
13 uRange = [3,6];
14 vRange = [-0.5, 1];
15
16 ##### #####
17 [Xu,Yu] = getPartialDerivative(X, Y, Z, u, v);
18
19 area = getArea(Xu,Yu, u, v, uRange, vRange);
20
21 [u,v] = meshgrid(uRange(1) : 0.1 : uRange(2), vRange(1) : 0.1 : vRange(2)
22   ↪ );

```

```

23
24 % sferra
25 X = cos(u) .* cos(v);
26 Y = sin(u) .* cos(v);
27 Z = sin(v);
28
29 % parabola
30 % X = v .* cos(u);
31 % Y = v .* sin(u);
32 % Z = v.^2;
33
34 surf(X,Y,Z)
35
36 function area = getArea(X_u, X_v,u,v, uRange, vRange)
37 E = X_u * X_u';
38 F = X_u * X_v';
39 G = X_v * X_v';
40
41 f = sqrt(E*G' - F*F');
42 fIntegralU = int(f, u, uRange(1), uRange(2));
43 fIntegralUV = int(fIntegralU, v, vRange(1), vRange(2));
44 area = fIntegralUV;
45 end
46
47
48 function [Xu,Xv] = getPartialDerivative(X, Y, Z, u, v)
49 Xu(1) = diff(X,u);
50 Xu(2) = diff(Y,u);
51 Xu(3) = diff(Z,u);
52
53 Xv(1) = diff(X,v);
54 Xv(2) = diff(Y,v);
55 Xv(3) = diff(Z,v);
56 end

```

area = 3*sin(1)

2.2 Superfici tensor product - Bezier

Scelti i gradi n e m ed un numero di coefficienti appropriato, disegnare la relativa superficie tensor product di Bezier.

Svolgimento

Ricordando che lo spazio tensor product $\Pi_{n,m}$ con i polinomi di Bernstein è

$$B_i^n(u)B_j^m(v) = \binom{n}{i}u^i(1-u)^{n-i}\binom{m}{j}v^j(1-v)^{m-j}$$

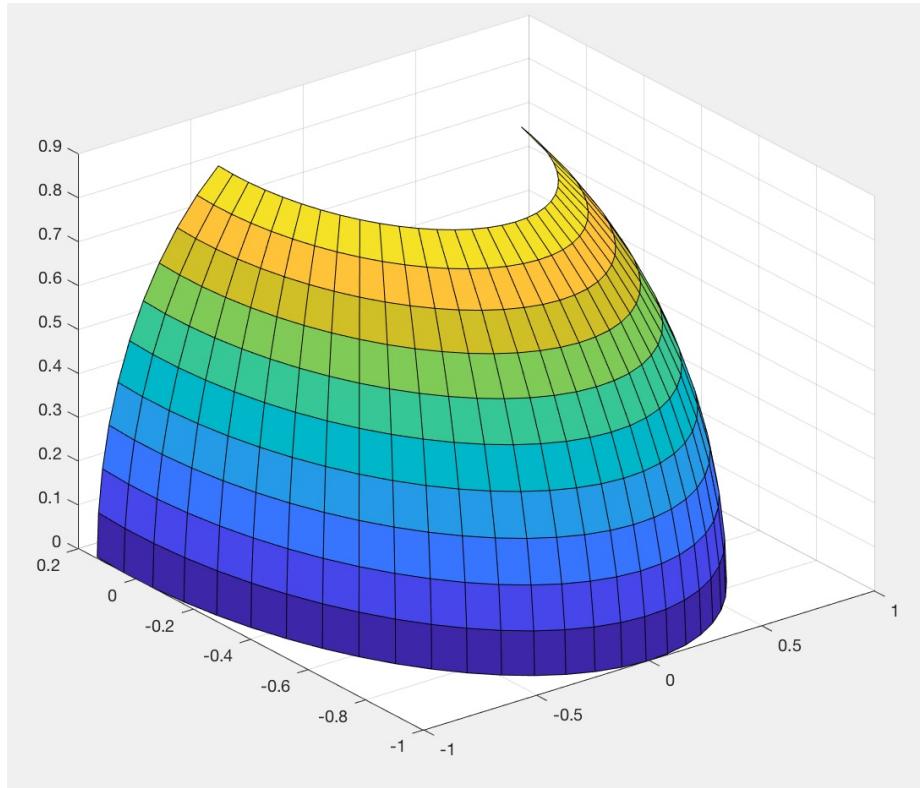


Figure 9: Rappresentazione di porzione di sfera

la superficie tensor product di Bezier è data dalla formula parametrica

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

dove i $b_{i,j}$, $i = 0, \dots, n$, $j = 0, \dots, m$ sono gli $(n+1)(m+1)$ punti di controllo della superficie.

L'implementazione dell'esercizio richiesto è quindi la seguente (output in figura 10):

```

1 n = 3;
2 m = 4;
3 %
4 ##########
5 xPoligon = 0:1/n:1;
6 yPoligon = 0:1/m:1;
7 zPoligon = rand(m+1,n+1)
8

```

```

9
10 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',...
11     ↪ 'LineWidth', 1,'LineStyle', '--')
12 [xValues,yValues,zValues] = getSurfacePoints(zPoligon);
13
14 hold on
15 surf(xValues,yValues, zValues)
16
17 title("Superficie tensor product Bezier con poligono di controllo")
18 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
19
20 function [x,y,z] = getSurfacePoints(coefs)
21 n = length(coefs(1,:))-1;
22 m = length(coefs(:,1))-1;
23 x = 0:1/100:1;
24 y = 0:1/100:1;
25 i_y = 1;
26 i_x = 1;
27 for u = 0:1/100:1
28     i_y = 1;
29     for v = 0:1/100:1
30         z(i_y,i_x) = getTensorBernsteinPoint(n,u,m,v, coefs);
31         i_y = i_y+1;
32     end
33     i_x = i_x+1;
34 end
35 end
36
37 function res = getTensorBernsteinPoint(n,u,m,v, coefs)
38 res = 0;
39 for i = 0:n
40     for j = 0:m
41         res = res + (coefs(j+1,i+1) * getBernsteinValue(n,i,u) *
42             ↪ getBernsteinValue(m,j,v));
43     end
44 end
45
46 function res = getBernsteinValue(k,i,u)
47 res = nchoosek(k,i) * u^i * (1-u)^(k-i);
48 end

```

2.3 Algoritmo di De Casteljau su superfici tensor product Bezier

Scelti i gradi n e m , i coefficienti opportuni e due valori (u,v) , rappresentare:

- il punto della relativa superficie tensor product di Bezier, corrispondente

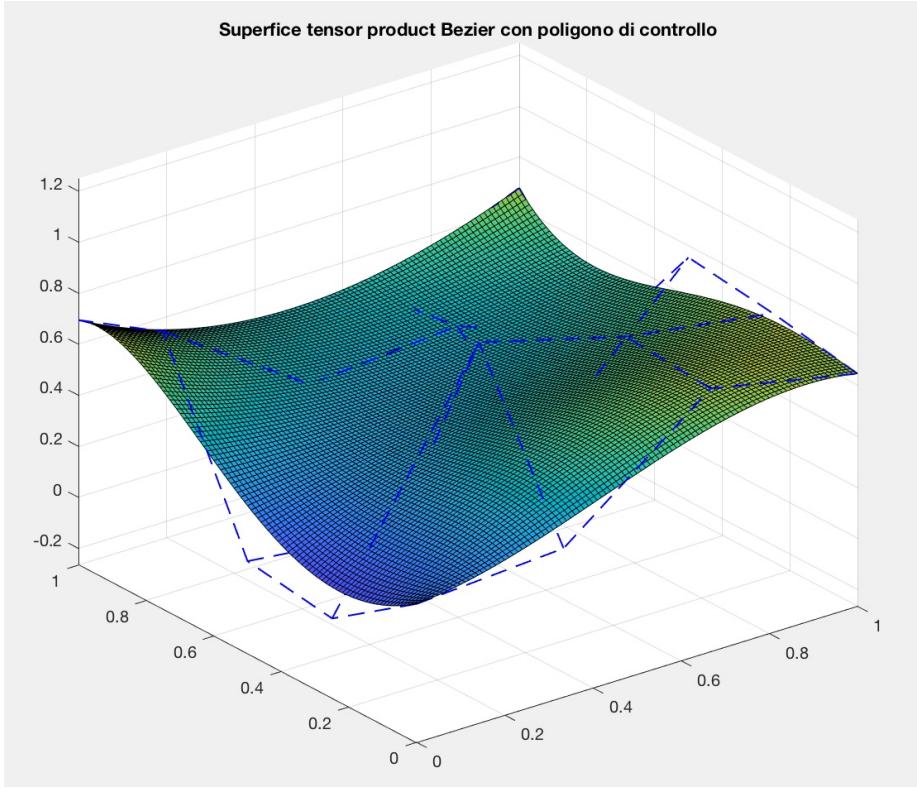


Figure 10: Rappresentazione della superficie tensor product di Bezier con poligono di controllo

ai valori (u, v) , utilizzando l'algoritmo di De Casteljau.

- l'intera superficie tensor product di Bezier utilizzando l'algoritmo di De Casteljau.

Svolgimento

Come per il caso bidimensionale anche nel caso tridimensionale è possibile ricavare ogni punto della superficie tensor product di Bezier attraverso l'applicazione ripetuta dell'algoritmo di De Casteljau sull'insieme dei coefficienti \mathbf{b} della curva.

Più precisamente, la formula da applicare (in versione matriciale) è la seguente

$$X(u, v) = b_{1,1}^{n,m}(u, v)$$

dove

$$b_{i,j}^{0,0}(u, v) = b_{i,j} \quad i = 0, \dots, n \quad j = 0, \dots, m$$

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

$r, s = 1, \dots, \gamma \quad i = 0, \dots, \gamma - r \quad j = 0, \dots, \gamma - s \quad \gamma = \min(r, s)$

Se $n \neq m$ allora si procede per γ passi col caso bivariato e per i restanti utilizzando l'algoritmo di de Casteljau nel caso univariato.

Un secondo approccio è quello di procedere prima in una direzione e poi in un'altra, ovvero

$$\begin{aligned} b_{i,j}^{r,s-1}(u,v) &= (1-u) \cdot b_{i,j}^{r-1,s-1}(u,v) + u \cdot b_{i+1,j}^{r-1,s-1}(u,v) \\ b_{i,j}^{r,s}(u,v) &= (1-v) \cdot b_{i,j}^{r,s-1}(u,v) + v \cdot b_{i,j+1}^{r,s-1}(u,v) \\ &= (1-v) \cdot \left[(1-u) \cdot b_{i,j}^{r-1,s-1}(u,v) + u \cdot b_{i+1,j}^{r-1,s-1}(u,v) \right] \\ &\quad + v \cdot \left[(1-u) \cdot b_{i,j+1}^{r-1,s-1}(u,v) + u \cdot b_{i+1,j+1}^{r-1,s-1}(u,v) \right] \end{aligned}$$

L'implementazione dell'esercizio, utilizzando il secondo approccio, è la seguente (output in figura 11):

```

1 n = 2; % grado base 1
2 m = 3; % grado base 2
3 u = 0.2; % coordinata u punto da calcolare
4 v = 0.3; % coordinata v punto da calcolare
5 %
6 ######
7
8 xPoligon = 0:1/n:1;
9 yPoligon = 0:1/m:1;
10 zPoligon = rand(m+1, n+1)
11 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',
12      'LineWidth', 1,'LineStyle','--')
13
14 z = getCastelJauPoint(1, 1, n, m, u, v, zPoligon);
15
16 [xValues,yValues,zValues] = getSurfacePoints(zPoligon);
17
18 hold on
19 surf(xValues,yValues, zValues,'FaceColor','none')
20
21 hold on
22 scatter3(u,v,z,'MarkerFaceColor','r')
23 title("punto trovato con algoritmo di De Casteljau")
24 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
25
26 ginput(1);
27 clf;
28 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',
      'LineWidth', 1,'LineStyle','--')
29 hold on

```

```

29 [xValues,yValues,zValues] = getSurfaceWithCasteljau(zPoligon);
30 surf(xValues,yValues, zValues,'FaceColor','r')
31 title("intera superficie trovata con algoritmo di De Casteljau")
32 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
33
34 function [xValues,yValues,zValues] = getSurfaceWithCasteljau(zPoligon)
35 n = length(zPoligon(1,:))-1;
36 m = length(zPoligon(:,1))-1;
37 idx = 1;
38 for u = linspace(0,1,100)
39     xValues(idx) = u;
40     idy = 1;
41     for v = linspace(0,1,100)
42         yValues(idy) = v;
43         zValues(idy,idx) = getCastelJauPoint(1,1,n,m,u,v,zPoligon);
44         idy = idy + 1;
45     end
46     idx = idx + 1;
47 end
48 end
49 function res = getCastelJauPoint(i, j, r, s, u, v, coefs)
50 if (r > 0)
51     res = (1-u) * getCastelJauPoint(i,j,r-1,s,u,v,coefs) + u *
52             ↪ getCastelJauPoint(i+1,j,r-1,s,u,v,coefs);
53 elseif (s > 0)
54     res = (1-v) * getCastelJauPoint(i,j,r,s-1,u,v,coefs) + v *
55             ↪ getCastelJauPoint(i,j+1,r,s-1,u,v,coefs);
56 else
57     res = coefs(j,i);
58 end
59 end

```

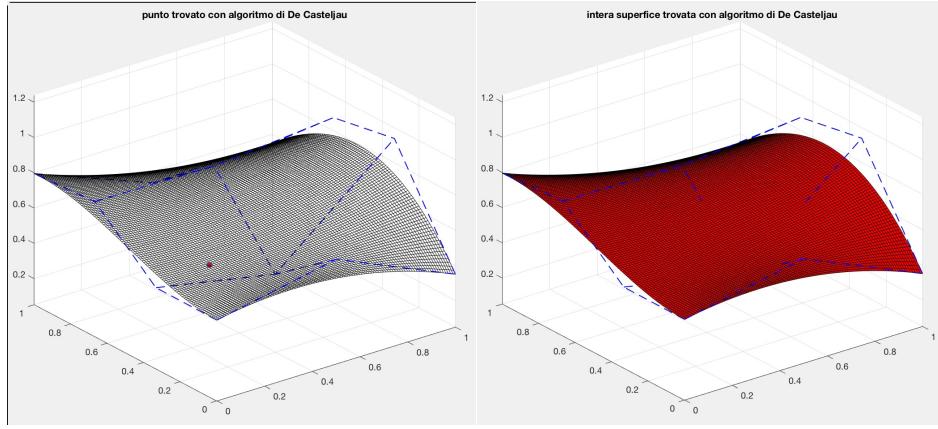
2.4 Superfici tensor product - B-Spline

Fissati gli ordini ku e kv delle funzioni B-Spline, i vettori dei nodi estesi e gli opportuni coefficienti, rappresentare:

- la relativa superficie tensor product B-Spline utilizzando la formula di definizione
- la relativa superficie tensor product B-Spline utilizzando il *Curve Fitting Toolbox* di Matlab.

Svolgimento

Siano $U = [u_0, \dots, u_{n+ku}]$, $V = [v_0, \dots, v_{m+kv}]$ i due vettori estesi di nodi associati agli intervalli $[a, b] = [u_{ku-1}, u_{n+1}]$, $[c, d] = [v_{kv-1}, v_{m+1}]$ allora la relativa



(a) Singolo punto

(b) Intera superficie

Figure 11: Algoritmo di de Casteljau

superficie tensor product B-Spline è data dalla seguente formula

$$\mathbf{X}(u, v) = \sum_{i=0}^n \sum_{j=0}^m \mathbf{d}_{i,j} N_{i,k_u}(u) N_{j,k_v}(v) \quad [u, v] \in [a, b] \times [c, d]$$

dove \mathbf{d} è la matrice degli $(n+1)(m+1)$ vertici di controllo della superficie.

L'implementazione di quanto richiesto è la seguente (output in figura 12):

```

1 kX = 3;
2 knotsX = augknt([0,1,2,3],kX);
3
4 kY = 4;
5 knotsY = augknt([1,2,3,4],kY);
6
7 % #####
8
9 xPoligon = linspace(knotsX(1),knotsX(end),length(knotsX)-kX);
10 yPoligon = linspace(knotsY(1),knotsY(end),length(knotsY)-kY);
11 zPoligon = rand(length(knotsY)-kY,length(knotsX)-kX);
12
13 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',...
14      'LineWidth', 1,'LineStyle','-.')
15 [xValues,yValues,zValues] = getSurfacePoints(zPoligon, knotsX, knotsY, kX
16      , kY);
17 hold on
18 surf(xValues,yValues, zValues)
19 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
20 title("Superficie con implementazione manuale")

```

```

21 ginput(1);
22 plotSurfacePointsWithhCurveFitting(kX,kY,knotsX,knotsY,zPoligon)
23 title("Superfice con curve fitting toolbox")
24
25 function plotSurfacePointsWithhCurveFitting(kX, kY, knotsX, knotsY, coefs
26     ↪ )
27 tabX = linspace(knotsX(1),knotsX(end),100);
28 tabY = linspace(knotsY(1),knotsY(end),100);
29 A = spcol(knotsX,kX,tabX);
30 B = spcol(knotsY,kY,tabY);
31
32 X = B*coefs*A';
33 [uu vv] = meshgrid(tabX,tabY);
34 surf(uu, vv, X,'EdgeColor','flat');
35 hold on
36 x = linspace(knotsX(1),knotsX(end),length(knotsX)-kX);
37 y = linspace(knotsY(1),knotsY(end),length(knotsY)-kY);
38 surf(x,y,coefs,'FaceColor','none','EdgeColor','b','LineWidth', 1,
39      ↪ LineStyle','-.')
40 end
41
42 function [x,y,z] = getSurfacePoints(coefs, knotsX, knotsY, kX , kY)
43 x = linspace(knotsX(1),knotsX(end),100);
44 y = linspace(knotsY(1),knotsY(end),100);
45
46 i_y = 1;
47 i_x = 1;
48 for u = x
49     i_y = 1;
50     for v = y
51         z(i_y,i_x) = getTensorBSplinePoint(kX,u,kY,v,coefs,knotsX, knotsY
52             ↪ );
53         i_y = i_y+1;
54     end
55     i_x = i_x+1;
56 end
57 end
58
59 function res = getTensorBSplinePoint(n,u,m,v, coefs, extendedKnots1,
60     ↪ extendedKnots2)
61 res = 0;
62 nc = length(coefs(1,:));
63 mc = length(coefs(:,1));
64 for i = 1:nc
65     nin = BSpline(i,n,u,extendedKnots1);
66     for j = 1:mc
67         njm = BSpline(j,m,v,extendedKnots2);
68         res = res + (coefs(j,i) * nin * njm );
69     end
70 end
71
72 end

```

67 | **end**

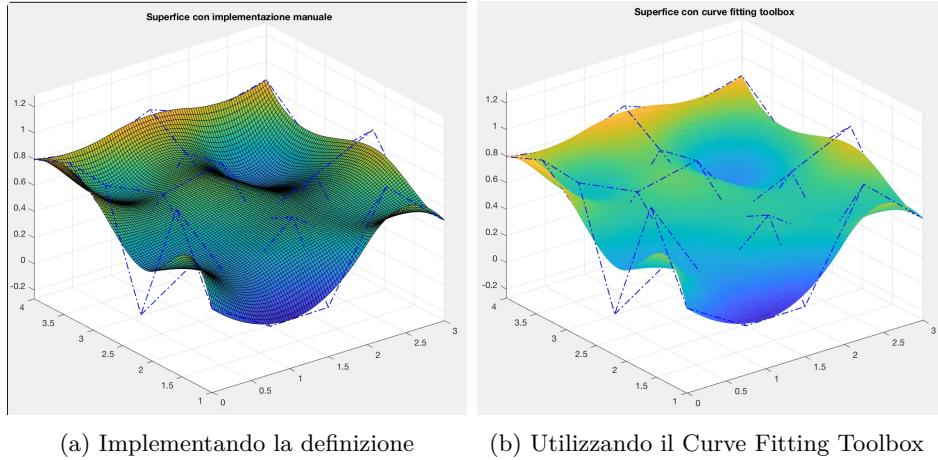


Figure 12: Superfici tensor product B-Spline

2.5 Algoritmo di De Boor su superfici tensor product B-Spline

Fissati gli ordini k e p delle funzioni B-Spline, i vettori dei nodi estesi, gli opportuni coefficienti e due valori (u, v) , rappresentare:

- il punto della relativa superficie tensor product B-Spline, corrispondente ai valori (u, v) , utilizzando l'algoritmo di De Boor.
- l'intera superficie utilizzando l'algoritmo di De Boor.

Svolgimento

Siano $U = [u_0, \dots, u_{n+k}]$, $V = [v_0, \dots, v_{m+p}]$ i due vettori estesi di nodi e $(u, v) \in [u_r, u_{r+1}] \times [v_s, v_{s+1}]$ allora come per il caso univariato possiamo riscrivere il punto $X(u, v)$ come segue (in forma matriciale):

$$X(u, v) = d_{r,s}^{(k-1,p-1)}(u, v)$$

dove

$$d_{i,j}^{(0,0)}(u, v) = d_{i,j} \quad i = 0, \dots, n \quad j = 0, \dots, m$$

$$d_{i,j}^{q,q}(u, v) = (1 - \omega_{i,k-q+1}(u), \omega_{i,k-q+1}(u)) \begin{pmatrix} d_{i-1,j-1}^{(q-1,q-1)}(u, v) & d_{i-1,j}^{(q-1,q-1)}(u, v) \\ d_{i,j-1}^{(q-1,q-1)}(u, v) & d_{i,j}^{(q-1,q-1)}(u, v) \end{pmatrix} \begin{pmatrix} 1 - \omega_{i,p-q+1}(v) \\ \omega_{i,k-p+1}(v) \end{pmatrix}$$

con $q = \min(k-1, p-1)$.

Se $k \neq p$ una volta eseguiti q passi dell'algoritmo si procede nella direzione rimanente fino a calcolare $d_{r,s}^{(k-1,p-1)}(u, v)$.

L'implementazione MatLab dell'esercizio richiesto è la seguente (output in figura 13):

```

1 kX = 4;
2 knotsX = augknt([0,1],kX);
3
4 kY = 3;
5 knotsY = augknt([1,2],kY);
6
7 u = 0.5;
8 v = 1.5;
9
10 #####%
11
12 xPoligon = linspace(knotsX(1),knotsX(end),length(knotsX)-kX);
13 yPoligon = linspace(knotsY(1),knotsY(end),length(knotsY)-kY);
14 zPoligon = rand(length(knotsY)-kY,length(knotsX)-kX);
15
16 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',...
17      ↪ 'LineWidth', 1,'LineStyle','-.')
18 hold on
19 plotSurfaceCurveFittingToobox(kX,kY,knotsX,knotsY,zPoligon);
20
21 z = getDeBoorPoint(u, v, knotsX, knotsY, kX, kY, zPoligon);
22
23 hold on
24 scatter3(u, v, z,'MarkerFaceColor','r');
25 title("punto trovato con algoritmo di De Boor")
26 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
27
28 ginput(1);
29 clf;
30 surf(xPoligon,yPoligon,zPoligon,'FaceColor','none','EdgeColor','b',...
31      ↪ 'LineWidth', 1,'LineStyle','-.')
32 hold on
33 [xValues,yValues, zValues] = getSurfaceWithDeBoor(knotsX, knotsY, kX, kY,
34      ↪ zPoligon);
35 surf(xValues,yValues, zValues,'FaceColor','r')
36 title("intera superficie con algoritmo di De Boor")
37
38 zlim([min(zPoligon(:))-0.3,max(zPoligon(:))+0.3]);
39
40 function [xValues,yValues, zValues] = getSurfaceWithDeBoor(knotsX, knotsY
41      ↪ , kX, kY, zPoligon)
42 idx = 1;
43 for u=linspace(knotsX(1),knotsX(end),100)
44     xValues(idx) = u;
45     idy = 1;
46     for v=linspace(knotsY(1),knotsY(end),100)
47         yValues(idy) = v;

```

```

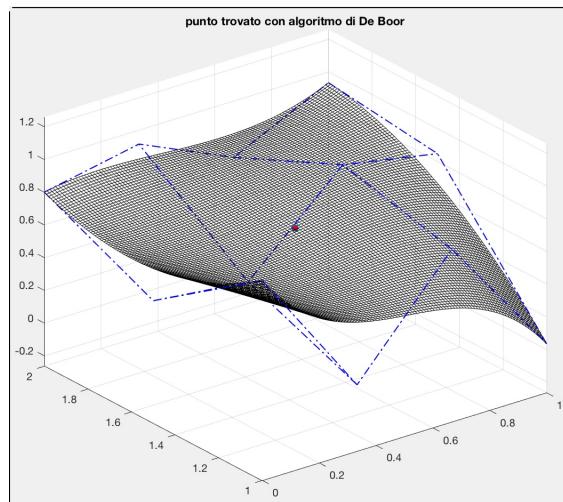
44     z = getDeBoorPoint(u, v, knotsX, knotsY, kX, kY, zPoligon);
45     zValues(idy, idx) = z;
46     idy = idy + 1;
47   end
48   idx = idx + 1;
49 end
50 end
51
52 function z = getDeBoorPoint(u, v, knotsX, knotsY, k, l, zPoligon)
53 r = find(knotsX < u);
54 if isempty(r) r = k; end
55 r = r(end);
56
57 s = find(knotsY < v);
58 if isempty(s) s = l; end
59 s = s(end);
60
61 z = getDerivative(r, s, u, v, k-1,l-1, knotsX, knotsY, zPoligon, k, l);
62 end
63
64 function res = getDerivative(i, j, u, v, r, s, knotsX, knotsY, zPoligon,
65   ↪ k, l)
65 if (r == 0 && s == 0)
66   res = zPoligon(j,i);
67 elseif (r ~= 0 && s ~= 0)
68   wikmqp1 = getW (i, k-r+1, u, knotsX);
69   wjlmqp1 = getW (j, l-s+1, v, knotsY);
70   dij = getDerivative(i, j, u, v, r-1, s-1, knotsX, knotsY, zPoligon, k
71   ↪ , l);
72   dim1jm1 = getDerivative(i-1, j-1, u, v, r-1, s-1, knotsX, knotsY,
73   ↪ zPoligon, k, l);
74   dim1j = getDerivative(i-1, j, u, v, r-1, s-1, knotsX, knotsY,
75   ↪ zPoligon, k, l);
76   dijm1 = getDerivative(i, j-1, u, v, r-1, s-1, knotsX, knotsY,
77   ↪ zPoligon, k, l);
78   res = [1-wikmqp1 wikmqp1] * [dim1jm1 dim1j ; dijm1 dij] * [1-wjlmqp1
79   ↪ wjlmqp1]';
80
81 elseif (r == 0 && s ~= 0)
82   wjlmqp1 = getW (j, l-s+1, v, knotsY);
83   dij = getDerivative(i, j, u, v, r, s-1, knotsX, knotsY, zPoligon, k,
84   ↪ l);
85   dijm1 = getDerivative(i, j-1, u, v, r, s-1, knotsX, knotsY, zPoligon,
86   ↪ k, l);
87
88   res = [dijm1 dij] * [1-wjlmqp1 wjlmqp1]';
89
90 elseif (r ~= 0 && s == 0)
91   wikmqp1 = getW (i, k-r+1, u, knotsX);

```

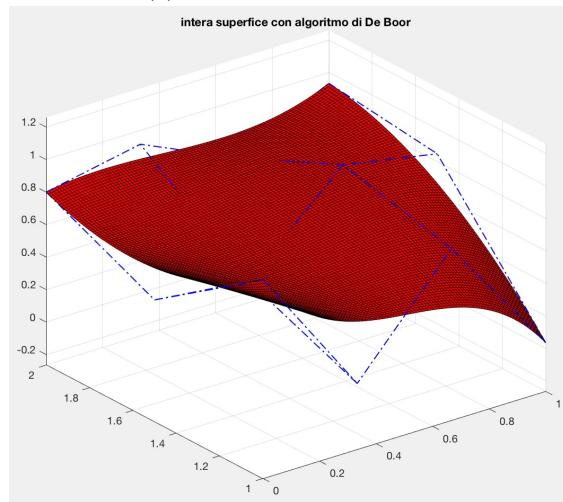
```

86     dij = getDerivative(i, j, u, v, r-1, s, knotsX, knotsY, zPoligon, k,
87     ↪ 1);
87     dim1j = getDerivative(i-1, j, u, v, r-1, s, knotsX, knotsY, zPoligon,
88     ↪ k, 1);
88
89     res = [1-wikmqp1 wikmqp1] * [dim1j ; dij];
90 end
91 end

```



(a) Singolo punto



(b) Intera superficie

Figure 13: Algoritmo di De Boor

3 Interpolazione

3.1 Interpolazione 2D

1. Scelti n punti sul piano e un ordine k , rappresentare la relativa curva di tipo B-Spline che intercala i punti scelti.
2. Mostrare una seconda curva interpolazione per gli stessi punti scelti

Svolgimento

Dati un vettore esteso dei nodi, un ordine k e un insieme di punti $\mathbf{f} = [f_1, \dots, f_n]$ possiamo definire il vettore delle *ascisse di interpolazione* \mathbf{t} tale che $s(t_i) = f_i$, $i = 1, \dots, n$ dove s rappresenta la spline di ordine k interpolante i punti dati. Definiamo inoltre la matrice \mathbf{A} detta *matrice di collocazione delle B-Spline* (nelle ascisse di interpolazione t_i) quella matrice ($n \times n$) tale che

$$\mathbf{A} = \begin{pmatrix} N_{1,k}(t_1) & \dots & N_{n,k}(t_1) \\ \vdots & \vdots & \vdots \\ N_{1,k}(t_n) & \dots & N_{n,k}(t_n) \end{pmatrix}.$$

A questo punto il problema si riduce nello scegliere le opportune ascisse di interpolazione \mathbf{t} per le quali la matrice \mathbf{A} risulti non singolare e di conseguenza il sistema

$$\mathbf{Ac} = \mathbf{f}$$

sia risolvibile, dove $\mathbf{c} = [c_1, \dots, c_n]^T$ rappresenta il vettore dei coefficienti della B-Spline interpolante i punti \mathbf{f} .

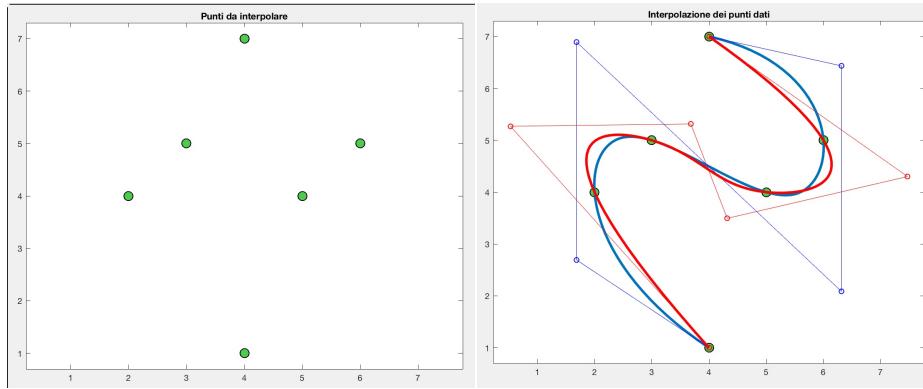
Qui di seguito l'implementazione dell'esercizio richiesto, utilizzando due diversi vettori di ascisse di interpolazione (output in figura 14):

```
1 f = [4 1; 2 4 ; 3 5; 5 4; 6 5; 4 7];
2 k = 4;
3 knots = augknt(0:length(f)-k+1,k);
4
5 % prima interpolazione
6 t = linspace(knots(1),knots(end),length(f));
7 A = spcol(knots, k, t);
8 coefs = A\f;
9 coefs = coefs';
10 X = spmak(knots,coefs);
11
12 % seconda interpolazione
13 t2 = aveknt(knots, k);
14 A2 = spcol(knots, k, t2);
15 coefs2 = A2\f;
16 coefs2 = coefs2';
17 X2 = spmak(knots,coefs2);
18
```

```

19 plot(f(:,1),f(:,2), 'ko', 'MarkerSize',11, 'MarkerFaceColor',[.3 .8 .3]);
20 title("Punti da interpolare")
21 xlim([min([coefs(1,:), coefs2(1,:)])-0.3, max([coefs(1,:),coefs2(1,:)])
22      ↪ +0.3]);
22 ylim([min([coefs(2,:), coefs2(2,:)])-0.3, max([coefs(2,:),coefs2(2,:)])
23      ↪ +0.3]);
23 ginput(1);
24 hold on;
25 fnplt(X,[knots(k),knots(end-k+1)],3);
26 title("Interpolazione dei punti dati")
27 plot(coefs(1,:),coefs(2,:),'b',coefs(1,:),coefs(2,:),'bo');
28 fnplt(X2,[knots(k),knots(end-k+1)],'r',3);
29 plot(coefs2(1,:),coefs2(2,:),'r',coefs2(1,:),coefs2(2,:),'ro');

```



(a) Punti scelti

(b) Interpolazioni proposte

Figure 14: Interpolazione 2D con curve B-Spline

3.2 Interpolazione 3D

Scelti $n \times m$ punti nello spazio e due ordini k e p , rappresentare la relativa superficie tensor product di tipo B-Spline che intercala i punti scelti.

Svolgimento

In maniera simile all'esercizio precedente, dobbiamo trovare le ascisse di interpolazione \mathbf{x} e \mathbf{y} tali che le due matrici di collocazione

$$\mathbf{A} = \begin{pmatrix} N_{1,k}(x_1) & \dots & N_{1,k}(x_n) \\ \vdots & \vdots & \vdots \\ N_{n,k}(x_1) & \dots & N_{n,k}(x_n) \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} N_{1,p}(y_1) & \dots & N_{1,p}(y_m) \\ \vdots & \vdots & \vdots \\ N_{m,p}(y_1) & \dots & N_{m,p}(y_m) \end{pmatrix}$$

siano non singolari ed il sistema

$$\mathbf{A}^T \mathbf{C} \mathbf{B} = \mathbf{F}$$

sia risolvibile, dove \mathbf{C} rappresenta la matrice dei coefficienti della superficie tensor product B-Spline interpolante la matrice \mathbf{F} dei punti dati.

Per risolvere tale sistema si può procedere in due fasi:

1. determinare la matrice \mathbf{D} tale che $\mathbf{DB} = \mathbf{F}$
2. determinare la matrice \mathbf{C} tale che $\mathbf{A}^T \mathbf{C} = \mathbf{D}$

L'implementazione dell'esercizio richiesto è la seguente (output in figura 15)

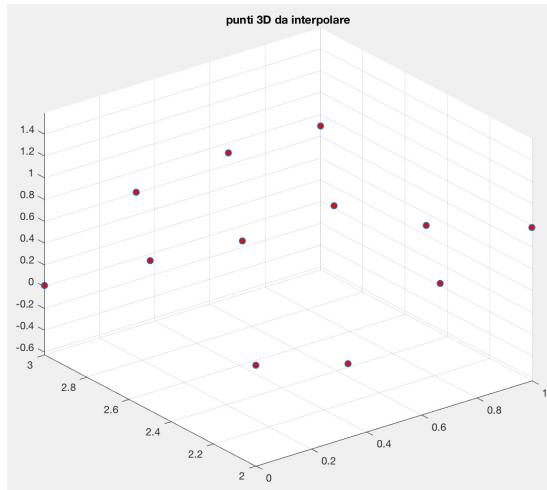
```

1 kX = 4;
2 knotsX = augknt([0,1],kX);
3
4 kY = 3;
5 knotsY = augknt([2,3],kY);
6
7 f =[0.2940 0.0484 0.5261 0.7814;
8     0.7463 0.6679 0.7297 0.2880;
9     0.0103 0.6035 0.7073 0.6925]
10
11 % f = rand(length(knotsY)-kY,length(knotsX)-kX)
12
13 x = linspace(knotsX(1),knotsX(end),length(knotsX)-kX);
14 y = linspace(knotsY(1),knotsY(end),length(knotsY)-kY);
15 xscatter = repmat(x, length(knotsY)-kY, 1);
16 xscatter = xscatter(:)';
17 yscatter = repmat(y, 1, length(knotsX)-kX);
18
19 A = spcol(knotsX, kX, x);
20 B = spcol(knotsY, kY, y);
21
22 D = f/A';
23 coefs = (D'/B')';
24
25 X = B*coefs*A';
26 [uu vv] = meshgrid(x,y);
27
28 scatter3(xscatter,yscatter,f(:, 'MarkerFaceColor', 'r');
29 title('punti 3D da interpolare');
30 zlim([min(coefs(:)),max(coefs(:))]);
31 ginput(1);
32 hold on
33 surf(uu, vv, coefs, 'FaceColor', 'none', 'EdgeColor', 'b', 'LineWidth', 1,
34     'LineStyle', '-.');
35 plotSurfaceCurveFittingToobox(kX, kY, knotsX, knotsY, coefs);
36 title('superficie interpolante i punti');
```

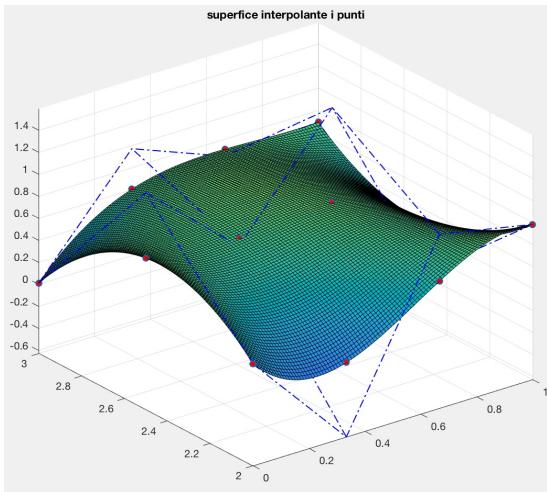
```

37 | function plotSurfaceCurveFittingToobox(k, l, knotsX, knotsY, coefs)
38 | tabX = linspace(knotsX(1), knotsX(end), 100);
39 | tabY = linspace(knotsY(1), knotsY(end), 100);
40 | A = spcol(knotsX,k,tabX);
41 | B = spcol(knotsY,l,tabY);
42 | X = B*coefs*A';
43 | [uu vv] = meshgrid(tabX,tabY);
44 | surf(uu, vv, X);
45 | end

```



(a) Punti scelti



(b) Interpolazioni proposte

Figure 15: Interpolazione 3D con superficie tensor product B-Spline

4 Hierarchical B-Spline

4.1 Basi HB-Spline

Considerando la definizione di HB-Spline, definire un opportuno insieme Ω di domini gerarchici e rappresentarne la base HB-Spline corrispondente.

Svolgimento

Siano B^0, \dots, B^{N-1} l'insieme delle basi B-Spline di livello $l = 0, \dots, N - 1$ tali che la molteplicità di ogni nodo al livello l sia minore o uguale alla molteplicità dello stesso nodo nel vettore dei nodi al livello successivo e sia $\Omega = \Omega^0, \dots, \Omega^{N-1}$ l'insieme dei domini gerarchici tali che $\Omega^0 \supseteq \Omega^1 \supseteq \dots \supseteq \Omega^{N-1}$ allora la base HB-Spline H è costruita secondo la seguente struttura ricorsiva:

(I) Inizializzazione: $H^0 = \{\beta \in B^0 : \text{supp } \beta \neq 0\}$

(II) Passo ricorsivo

$$H^{l+1} = H_A^{l+1} \cup H_B^{l+1} \quad l = 0, \dots, N - 2$$

dove

$$H_A^{l+1} = \{\beta \in H^l : \text{supp } \beta \not\subseteq \Omega^{l+1}\} \quad H_B^{l+1} = \{\beta \in B^{l+1} : \text{supp } \beta \subseteq \Omega^{l+1}\}$$

(III) $H = H^{N-1}$

L'implementazione dell'esercizio è quindi la seguente (output in figura 16):

```

1 k = 4;
2 knots1 = augknt(0:6:54,k);
3 knots2 = augknt(0:3:54,k);
4 knots3 = augknt(0:2:54,k);
5
6 % level extended knots
7 omegas{1}{1}={knots1};
8 omegas{2}{1}={knots2};
9 omegas{3}{1}={knots3};
10
11 % level domain ranges
12 omegas{1}{2}={[0,72]};
13 omegas{2}{2}={[12,36]};
14 omegas{3}{2}={[18,30]};
15 % #####
16 plotRange = [knots1(1), knots1(end)];
17 color1 = [0 0 0]
18 color2 = [.9 .2 .5]
19 color3 = [.0 .5 .9]
```

```

22 subplot(4,2,1);
23 plotAllBasesBSpline(knots1,k,plotRange, color1);
24 title('B-Spline liv.0')
25 subplot(4,2,3);
26 plotAllBasesBSpline(knots2,k,plotRange, color2);
27 title('B-Spline liv.1')
28 subplot(4,2,5);
29 plotAllBasesBSpline(knots3,k,plotRange, color3);
30 title('B-Spline liv.2')
31
32 HBSbases1 = getHierachicalBSplineBases4Level(omegas,k,1);
33 HBSbases2 = getHierachicalBSplineBases4Level(omegas,k,2);
34 HBSbases3 = getHierachicalBSplineBases4Level(omegas,k,3);
35
36 subplot(4,2,2);
37 plotBasesBSpline(knots1, k, HBSbases1, plotRange, color1);
38 title('HB-Spline liv.0')
39 subplot(4,2,4);
40 plotBasesBSpline(knots2, k, HBSbases2, plotRange, color2);
41 title('HB-Spline liv.1')
42 subplot(4,2,6);
43 plotBasesBSpline(knots3, k, HBSbases3, plotRange, color3);
44 title('HB-Spline liv.2')
45
46 subplot(4,2,8);
47 hold on
48 plotBasesBSpline(knots1, k, HBSbases1, plotRange, color1);
49 hold on
50 plotBasesBSpline(knots2, k, HBSbases2, plotRange, color2);
51 hold on
52 plotBasesBSpline(knots3, k, HBSbases3, plotRange, color3);
53 title('Base HB-Spline')
54
55 function bases = getHierachicalBSplineBases4Level(omegas,k,level)
56 bases = [];
57 knotsLevel = omegas{level}{1}{1};
58 rangeLevel = omegas{level}{2}{1};
59 if(level < length(omegas))
60     rangeNextLevel = omegas{level+1}{2}{1};
61 else
62     rangeNextLevel = [inf,inf];
63 end
64
65 M = length(knotsLevel)-k*2;
66 spaceDimension = k+M;
67 for i = 1:spaceDimension
68     if (knotsLevel(i) < rangeLevel(1) || knotsLevel(i+k-1) > rangeLevel
69         ↪ (2) )
70         continue;
    elseif (knotsLevel(i) < rangeNextLevel(1) || knotsLevel(i+k-1) >

```

```

71      ↪ rangeNextLevel(2))
72      bases = [bases, i];
73  end
74 end

```

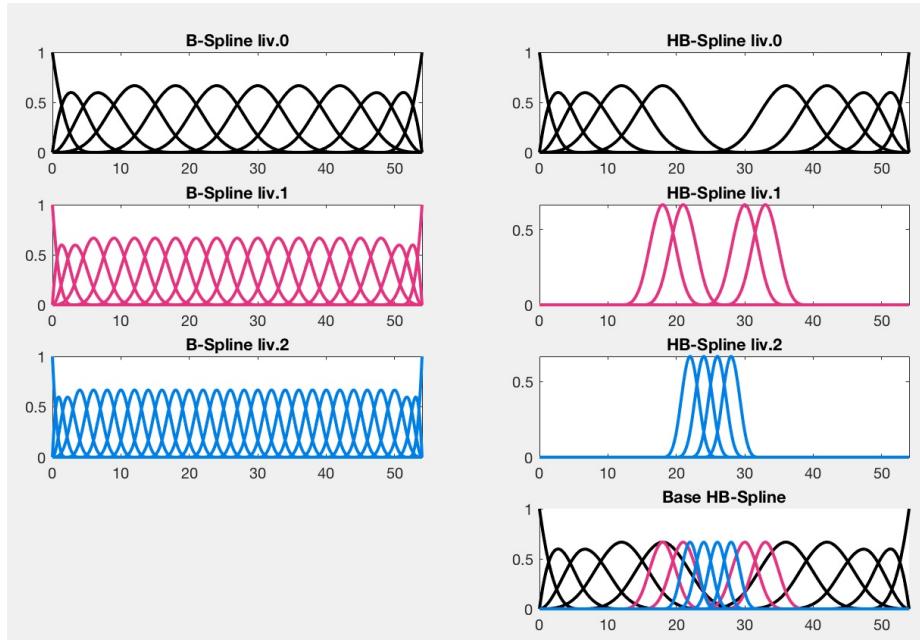


Figure 16: Costruzione della base HB-Spline (a destra) partendo dall’insieme delle basi B-Spline classiche (a sinistra).