Metodi Numerici per la Grafica

- Relazione -

Superfici di Bezier e B-Spline

Marco Calamai - Elia Mercatanti

23 novembre 2019

Indice

Listings

Elenco delle figure

1 Base delle B-spline

Sia assegnato $I = [\tau_0, \tau_L]$ e dato un vettore esteso di nodi

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

con

$$t_0 \leq t_1 \leq \ldots t_{k+1} \leq t_k \cdots \leq t_{n+1} \leq t_{n+2} \leq \cdots \leq t_{n+k}$$

in cui ogni nodo τ_i è ripetuto con molteplicità m_i , $i = 1, \ldots, L-1$.
Possiamo definire la base delle B-spline come l'insieme delle funzioni B-spline definite sul vettore esteso di nodi dalla formula ricorrente di *Cox - De Boor*

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

con

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

dove

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

Nel seguente codice Matlab 1 sono state implementate le funzioni descritte sopra per il calcolo delle basi di *Cox - De Boor*.
La funzione principale è `cox_deBoor`, la quale si occupa di calcolare le basi delle B-Spline richiamando la funzione `omega` per il calcolo dei coefficienti $\omega_{i,r}$ nella relazione ricorrente di *Cox - De Boor*.
A livello implementativo, nella funzione `cox_deBoor` la parte più delicata è il controllo da effettuare nel caso in cui l'ordine della B-Spline sia 1. In quel caso la funzione restituisce 1 se il valore di t_star cade nell'intervallo $[t_i, t_{i+1})$ ma nel caso in cui si trovasse nell'ultimo intervallo, bisogna prendere in considerazione anche l'ultimo valore dell'intervallo.
Per quanto riguarda invece la funzione `omega`, è stato previsto un controllo per fare in modo che restituisca zero nel caso di denominatore nullo, che si può verificare in caso di nodi con molteplicità maggiore di uno.

Listing 1: B-Spline tramite relazione ricorrente di Cox - de Boor

```
1 function [y] = cox_de_boor(i, r, order, knot_vector, t_star)
2 % COX_DE_BOOR:
3 %   Implementation of Cox-de Boor recursion formula for evaluating
4 %   B-Spline basis, return the evaluation of i-th element of the B
    -Spline
```

```
5  %   base of a given order and a knot vector in a t_star parameter
       value.
6  %
7  % Syntax: [y] = cox_de_boor(i, r, order, knot_vector, t_star)
8  %
9  % Input:
10 %   - i: i-th element of the B-Spline base.
11 %   - r: recursive order of the B-Spline base (used in recursion
       formula).
12 %   - order: real order of the B-Spline base.
13 %   - knot_vector: knot vector of the B-Spline curve.
14 %   - t_star: parameter value in which i want to evaluate the base
        element.
15 %
16 % Output:
17 %   - y: estimated value of the i-th element of the base
       corresponding to
18 %        t_star.
19 %
20 % Example:
21 %   knot_vector = [0 0 0 1 1 1];
22 %   y = cox_de_boor(1, 4, 4, knot_vector, 0.5);
23
24 % Authors: Elia Mercatanti, Marco Calamai
25 % Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
       it
26
27     % Check if we have arrived at the base case
28     if r == 1
29         % Check if t_star is included in the proper interval or in
               the last
30         % interval special case.
31         if (t_star >= knot_vector(i) && t_star < knot_vector(i+1))
               || ...
32             ((t_star >= knot_vector(i) && t_star <= knot_vector(i
                   +1) && ...
33             t_star == knot_vector(end) && i == length(knot_vector)-
                   order))
34             y = 1;
35         else
36             y = 0;
37         end
38     else
39         % Main Cox-de Boor recursion formula.
40         y = omega(i, r, knot_vector, t_star)*cox_de_boor(i, r-1,
               ...
41             order, knot_vector, t_star) + (1 - omega(i+1, r, ...
42             knot_vector, t_star)) * cox_de_boor(i+1, r-1, order,
                   ...
```

```
43              knot_vector, t_star);
44      end
45  end
46
47  function [omega_ir] = omega(i, r, knot_vector, t_star)
48  % OMEGA:
49  %   Calculate the coefficients omega_ir in the Cox-de Boor
        recursion
50  %   formula.
51  %
52  % Syntax: [omega_ir] = omega(i, r, knot_vector, t_star)
53  %
54  % Input:
55  %   - i: i-th element of the B-Spline base.
56  %   - r: recursive order of the B-Spline base (used in recursion
        formula).
57  %   - knot_vector: knot vector of the B-Spline curve.
58  %   - t_star: parameter value in which i want to evaluate the base
        element.
59  %
60  % Output:
61  %   - omega_ir: returned value of coefficients omega_ir.
62  %
63  % Example:
64  %   knot_vector = [0 0 0 1 1 1];
65  %   omega_ir = omega(1, 2, knot_vector, 0.5);
66
67  % Authors: Elia Mercatanti, Marco Calamai
68  % Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
        it
69
70      % Main calculations for the coefficient and denominator check
            (~= 0).
71      if t_star <= knot_vector(i+r-1) && knot_vector(i+r-1) ~=
            knot_vector(i)
72          omega_ir = (t_star-knot_vector(i)) / (knot_vector(i+r-1) -
                ...
73                  knot_vector(i));
74      else
75          omega_ir = 0;
76      end
77  end
```

Per disegnare degli esempi di basi B-Spline abbiamo utilizzato lo sript 2 in cui viene richiamata la funzione 1 per il calcolo delle basi di *Cox - De Boor* vista prima.
In figura 1 sono riportate le sei funzioni di un'esempio di base di Bernstein di ordine 6, la quale rappresenta un caso particolare di B-Spline in cui, il vettore esteso dei

nodi è formato solamente da τ_0 ripetuto ordine volte, seguito da τ_L ripetuto ordine volte.
La figura 2 invece mostra un esempio di base B-SPline di ordine 4 ($k = 4$) con vettore esteso di nodi definito come $t = [1, 1, 2, 3, 4, 5, 6, 6]$.

Listing 2: Disegno basi B-Spline

```
% B_SPLINE_BASE:
%   Takes as input the order and a knot vector to evaluate and plot all
%   elements of the relative B-Spline base, using Cox-de Boor recursion
%   formula.
%
% Requires:
%   - cox_de_boor.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.it

clear

% Ask user for inputs.
prompt = {'Order:', 'Knocts Vector:'};
inputs_title = 'Insert Inputs';
dimensions = [1 50];
default_inputs = {'3', '[0 0 0 1 1 1]'};
inputs = inputdlg(prompt, inputs_title, dimensions, default_inputs);

% Retrive inputs.
order = str2double(inputs{1});
knot_vector = str2num(inputs{2});
num_curve_points = 1000;

% Initialize steps and vector of evaluations for drawing base curve.
steps = linspace(knot_vector(1), knot_vector(end), num_curve_points);
base_element = zeros(1, num_curve_points);

% Set the figure window for drawing plots.
fig = figure('Name', 'B-Spline Base', 'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
```

```
ylabel('Y');
title('B-Spline Base');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';

% Calculate and plot curves for the elements of the base using
% Cox-de Boor recursion formula.
for i = 1 : length(knot_vector) - order
    for j = 1 : num_curve_points
        base_element(j) = cox_de_boor(i, order, order, knot_vector
            , ...
                                      steps(j));
    end
    ordinal = iptnum2ordinal(i);
    plot(steps, base_element, 'linewidth', 3, 'DisplayName', ...
         [upper(ordinal(1)), ordinal(2:min(end)) ' Base Element'])
            ;
end
legend('Location', 'best');
```

2 Le Curve B-spline

Dati $n+1$ punti di controllo, un curva B-Spline $\mathbf{X} : [a,b] = [\tau_0, \tau_L]$ di ordine k è definita a partire dalla base delle B-Spline come

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

Le curve B-Spline presenti in questa relazione sono state calcolate e rappresentate utilizzando l'algoritmo di De Boor.
L'algoritmo di De Boor è una generalizzazione dell'algoritmo di De Casteljeau; un modo veloce e numericamente stabile per trovare un punto in una B-Spline dato un **u** appartenente al dominio. Tale algoritmo si basa sul fatto che aumentando la molteplicità di un knot interno, decresce il numero di funzioni base non nulle che attraversano questo knot, infatti, se la molteplicità di questo knot è m, ci sono al più $degree \breve{} m + 1$ funzioni base non nulle che attraversano questo knot. Questo implica che in un nodo di molteplicità pari al grado della curva, ci sarà solo un funzione base (essendo $degree \breve{} degree + 1 = 1$) non nulla il cui valore in corrispondenza di tale knot sarà uguale ad 1 per il principio della partizione dell'unità. Quindi, nell'algoritmo di De Boor, un nodo u viene inserito ripetutamente in modo che la sua molteplicità sia pari al grado della curva. L'ultimo nuovo punto di controllo generato sarà quindi il punto della curva che corrisponde ad u.

Figura 1: Esempio di base con t = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] e k=6

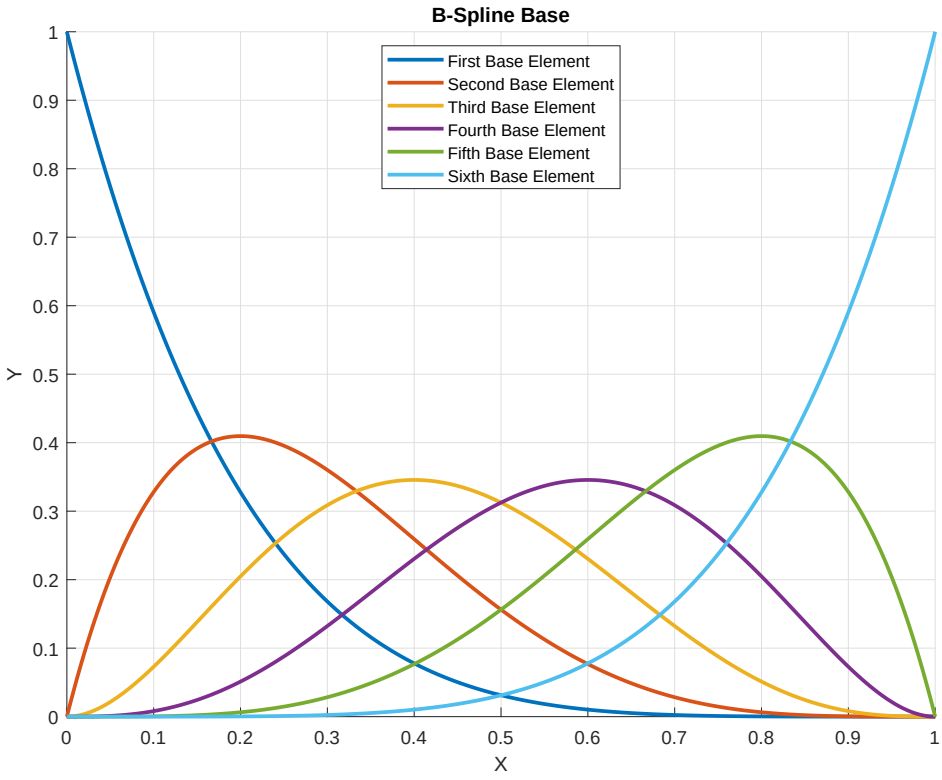
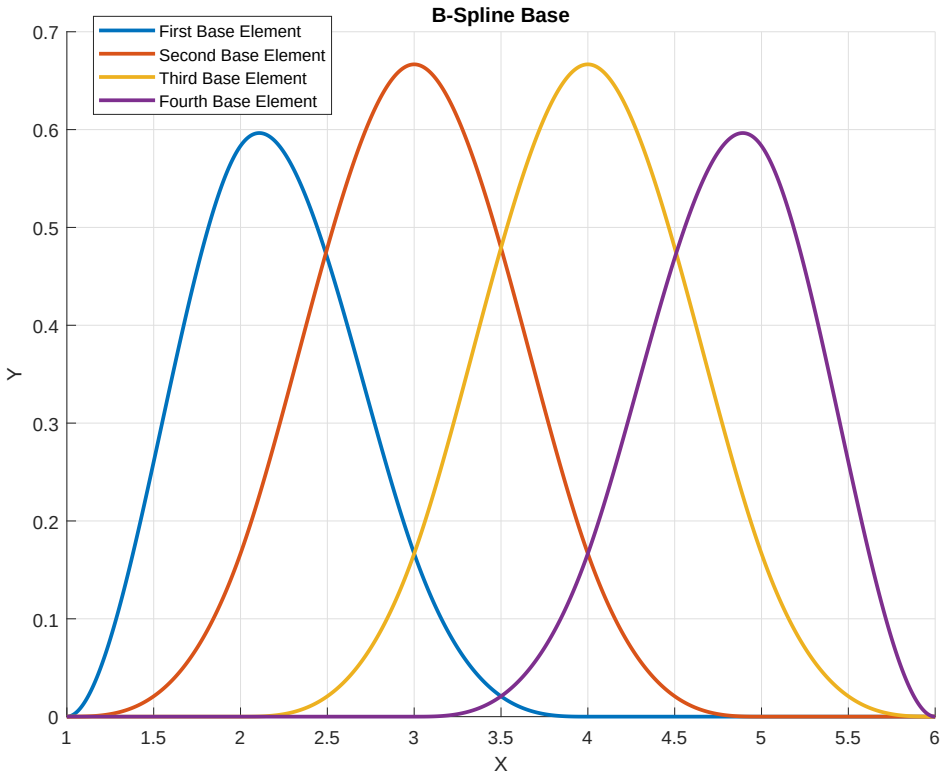


Figura 2: Esempio di base con t = [1, 1, 2, 3, 4, 5, 6, 6] e k= 4



Nella funzione Matlab 3 è stato implementato l'algoritmo di De Boor per il calcolo e la rappresentazione di curve B-Spline. La descrizione dell'algoritmo è illustrata nello pseudo codice 16.

Algorithm 1 Algoritmo di De Boor

Input: $\mathbf{u}$
Output: $\mathbf{C}(\mathbf{u})$, il valore della curva in $\mathbf{u}$.
  if $\mathbf{u}$ non è un nodo già esistente **then**
    $h = degree$ (inseriamo $\mathbf{u}$ esattamente grado volte)
    $s = 0$
  else
    $h = degree - s$ (inseriamo $\mathbf{u}$ $degree - s$ volte, dove s è la molteplicità del nodo già esistente)
  end if
  Supponiamo che il nodo $\mathbf{u}$ si trovi nel knot span $[\mathbf{u}_l, \mathbf{u}_{l+1})$.
  Copiamo i punti di controllo che saranno influenzati dall'algoritmo $\mathbf{P}_{l-s}, \mathbf{P}_{l-s-1}, \mathbf{P}_{l-s-2}, \ldots, \mathbf{P}_{l-degree+1}, \mathbf{P}_{l-degree}$ in un nuovo array e li rinominiamo come: $\mathbf{P}_{l-s,0}, \mathbf{P}_{l-s-1,0}, \mathbf{P}_{l-s-2,0}, ..., \mathbf{P}_{l-degree+1,0}, \mathbf{P}_{l-degree,0}$ dove lo 0 indica lo step iniziale (che ovviamente crescerà ad ogni passo dell'algoritmo);
  for r from 1 to h **do**
    text
    for i from $l - degree + r$ to $l - s$ **do**
      $\mathbf{a}_{i,r} = \frac{\mathbf{u} - \mathbf{u}_i}{\mathbf{u}_{i+degree-r+1} - \mathbf{u}_i}$
      $\mathbf{P}_{i,r} = (1 - \mathbf{a}_{i,r}) * \mathbf{P}_{i-1,r-1} + \mathbf{P}_{i,r-1}$
    end for
  end for
  return $\mathbf{P}_{l-s,degree-s}$

Listing 3: Algoritmo di De Boor

```
1  function [curve_point] = de_boor_algorithm(control_points, degree,
       ...
2                                                knot_vector, t_star)
3
4      % Find index of knot interval that contains t_star.
5      k = find(knot_vector <= t_star);
6      k = k(end);
7
8      % Calculate the multiplicity of t_star in t (0 <= s <= degree
         +1).
9      s = sum(eq(t_star, knot_vector));
10
11     % Num. times t_star must be repeated to reach a multiplicity
```

```
12      % equal to the degree.
13      h = degree - s;
14
15      % Copy of influenced control points.
16      order = degree + 1;
17      P_ir = zeros(order, size(control_points, 2), order);
18      P_ir((k-degree):(k-s), :, 1) = control_points((k-degree):(k-s)
            , :);
19
20      % Main De Boor algorithm.
21      q = k - 1;
22      if h > 0
23          for r = 1:h
24              for i = q-degree+r : q-s
25                  a_ir = (t_star - knot_vector(i+1)) / (knot_vector(
                        i+ ...
26                              degree-r+2)-knot_vector(i+1));
27                  P_ir(i+1, :, r+1) = (1 - a_ir)*P_ir(i, :, r) +
                        a_ir * ...
28                                          P_ir(i+1, :, r);
29              end
30          end
31          curve_point = P_ir(k-s, :, h+1);
32      elseif k == numel(knot_vector)
33          curve_point = control_points(end, :);
34      else
35          curve_point = control_points(k-degree, :);
36      end
37  end
```

2.1 Proprietà

Le principali proprietà delle curve B-Spline sono le seguenti:

Invarianza per Trasformazioni Affini. La proprietà di essere una partizione dell'unità garantisce che le curve B-spline siano invarianti per trasformazioni affini, ovvero queste due procedure producono lo stesso risultato:
Dati i vertici di controllo:

- Calcolo la curva e poi le applico la trasformazione affine.
- Applico la trasformazione affine ai vertici di controllo e poi calcolo la curva.

L'importanza pratica di questa proprietà è la seguente. Supponiamo di aver disegnato una curva e di volerle applicare una certa trasformazione affine (rotazione, traslazione, scala, ...). Il modo più semplice di procedere è applicare ai vertici di controllo la trasformazione affine desiderata, poi ridisegnare la curva.

Listing 4: Trasformazioni affini - Traslazione Rotazione e Scalatura

```
% B_SPLINE_CURVE_AFFINE_TRANS:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.it

clear

% Retrive inputs.
x_left_limit = 0;
x_right_limit = 1;
y_left_limit = 0;
y_right_limit = 1;
num_curve_points = 1000;
knot_vector = [0 0 0 0.3 0.6 1 1 1];
degree = 2;
control_points = [0.1 0.6; 0.3 0.9; 0.7 0.8; 0.8 0.5; 0.4 0.6];

% Set the figure window for drawing plots.
fig = figure('Name', 'B-spline Affine Transformations', 'NumberTitle', ...
             'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
ylabel('Y');
title(['B-spline Affine Transformations - Translation, Rotation and', ...
       ' Scaling']);
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([x_left_limit x_right_limit]);
ylim([y_left_limit y_right_limit]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(knot_vector(degree+1), knot_vector(end-degree), ...
                 num_curve_points);

% Plot control points and control polygon.
```

```
poi_plot = plot(control_points(:, 1), control_points(:, 2), 'k.', ...
                'MarkerSize', 20);
pol_plot = plot(control_points(:, 1), control_points(:, 2), '-', ...
                'linewidth', 1, 'color', '#0072BD');

% Calculate and plot the original B-Spline curve using De Boor algorithm.
curve = zeros(num_curve_points, 2);
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
original_curve = curve;
original_curve_plot = plot(curve(:, 1), curve(:, 2), 'linewidth', 3, ...
                           'color', '#D95319');

% Tranlation, rotation and scaling transformations.
translation = [0.9 1];
rotation = [cos(pi) -sin(pi); sin(pi) cos(pi)];
scaling = [0.8 0; 0 0.8];

% Transformation on control points.
control_points = (control_points*rotation + translation)*scaling;

% Plot transformed control points and control polygon.
plot(control_points(:, 1), control_points(:, 2), 'k.', 'MarkerSize', 20);
plot(control_points(:, 1), control_points(:, 2), '-', 'linewidth', 1, ...
     'color', '#0072BD');

% Calculate and plot the transformed B-Spline curve on control points.
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
trasf_control_plot = plot(curve(:, 1), curve(:, 2), 'linewidth', 3, ...
                          'color', '#EDB120');

% Plot transformation on B-Spline curve points and legend.
original_curve = (original_curve*rotation + translation)*scaling;
trasf_curve_plot = plot(original_curve(:, 1), original_curve(:, 2), ...
                          '--', 'linewidth', 3, 'color', '#7E2F8E')
```

```
                                ;
legend([poi_plot pol_plot original_curve_plot ...
        trasf_control_plot trasf_curve_plot], 'Control Points',
           ...
        'Control Polygons', 'Original B-Spline Curve', ...
        'Transformations on Control Points', 'Transformations on
           Curve',...
        'Location', 'southeast');
```

Nello Script Matlab 4 è stata inizialmente definita e disegnata una curva B-Spline e successivamente applicata una trasformazione affine prima ai suoi punti di controllo e successivamente alla curva. In particolare sono state applicate in quest'ordine una rotazione traslazione e scalatura, inizialmente ai vertici di controllo originali, ottenendo la curva di coloro arancione mostrata in Figura 3. Successivamente sono state applicate le stesse trasformazioni direttamente sulla curva originale ottenendo la B-Spline di colore viola mostrata in Figura 3. La proprietà di invarianza per trasformazioni affini viene confermata dal fatto che le due curve combaciano.

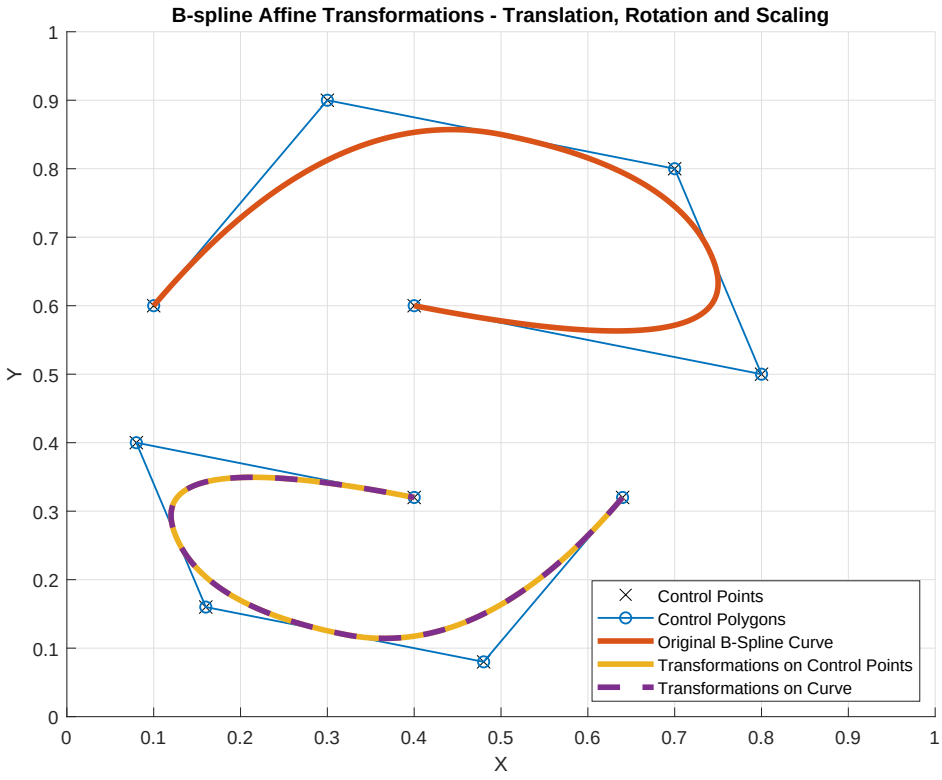
Località. Muovendo $\mathbf{d_i}$ la curva $\mathbf{X}(t)$ cambia solo nell'intervallo $[t_i; t_{i+k})$. Questo segue dal fatto che $N_{i,k}(t) = 0$ per $t \notin [t_i; t_{i+k})$. Equivalentemente, $\mathbf{d_i}$ influenza solo al più k segmenti di curva.

Negli script 5 e 6 viene mostrata la proprietà di località. Nel primo script 5 viene mostrata come descritta sopra, ovvero data una B-Spline con 10 punti di controllo, spostando il quinto punto la curva varia solamente nell'intervallo $[t_5, t_9]$. Questo comportamento lo si può vedere in Figura 4. 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 $d_{e-k+1}, \ldots, d_r$. Nello script 6 è mostrato questo comportamento, scegliendo $r = 6$ e modificando i punti di controllo $d_j \notin [d_3, d_6]$ la curva $\mathbf{X}(t^*)$ rimane invariata per $t^* \in [t_6, t_7)$ come si può vedere in Figura 5.

Listing 5: Proprietà di Località

```
% B_SPLINE_LOCALITY:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
   it

clear

% Retrive inputs.
```

Figura 3: Trasformazioni Affini di una Curva B-spline



```
x_left_limit = 0;
x_right_limit = 1;
y_left_limit = 0;
y_right_limit = 1;
num_curve_points = 1000;
knot_vector = [0 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1 1];
degree = 3;
control_points = [0.1, 0.1; 0.3, 0.4; 0.1, 0.6; 0.3, 0.9; 0.5, 0.3; ...
                  0.8, 0.9; 0.9, 0.6; 0.9, 0.3; 0.8, 0.2; 0.7, 0.1];

% Set the figure window for drawing plots.
fig = figure('Name', 'Locality Property 1', 'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
ylabel('Y');
title('Locality Property 1');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([x_left_limit x_right_limit]);
ylim([y_left_limit y_right_limit]);

% Get the order of the B-Spline curve.
order = degree + 1;

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(knot_vector(order), knot_vector(end-degree), ...
                 num_curve_points);

% Plot control points and control polygon of the original curve.
plot(control_points(:, 1), control_points(:, 2), 'kx', 'MarkerSize', 10);
plot(control_points(:, 1), control_points(:, 2), '-', 'linewidth', 1, ...
     'color', '#0072BD');

% Calculate and plot the original B-Spline curve using De Boor algorithm.
curve = zeros(num_curve_points, 2);
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
```

```
plot(curve(:, 1), curve(:, 2), 'linewidth', 3, 'color', '#D95319');

% Control point modification.
control_point_mod = 5;
control_points(control_point_mod, :) = [0.5 0.6];

% Plot control points and control polygon of the modified curve.
plot(control_points(:, 1), control_points(:, 2), '-.o', 'linewidth', 1, ...
     'color', '#EDB120', 'MarkerEdgeColor', 'k', 'MarkerSize', 10);

% Calculate and plot the modified B-Spline curve.
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
plot(curve(:, 1), curve(:, 2), '--', 'linewidth', 3, 'color', '#77AC30');

% Plot lines for highlighting the modified interval.
left_line = de_boor_algorithm(control_points, degree, knot_vector, ...
                              knot_vector(control_point_mod));
right_line = de_boor_algorithm(control_points, degree, knot_vector, ...
                               knot_vector(control_point_mod+order));
plot([left_line(1) left_line(1)], [y_left_limit y_right_limit], 'k', ...
     'linewidth', 2)
plot([right_line(1) right_line(1)], [y_left_limit y_right_limit], 'k', ...
     'linewidth', 2)
legend({'Control Points', 'Original Control Polygon', ...
        'Original B-Spline Curve', 'Modified Control Polygon', ...
        'Modified B-Spline Curve', 'Sector of Change'}, 'Location', ...
        'south');
```

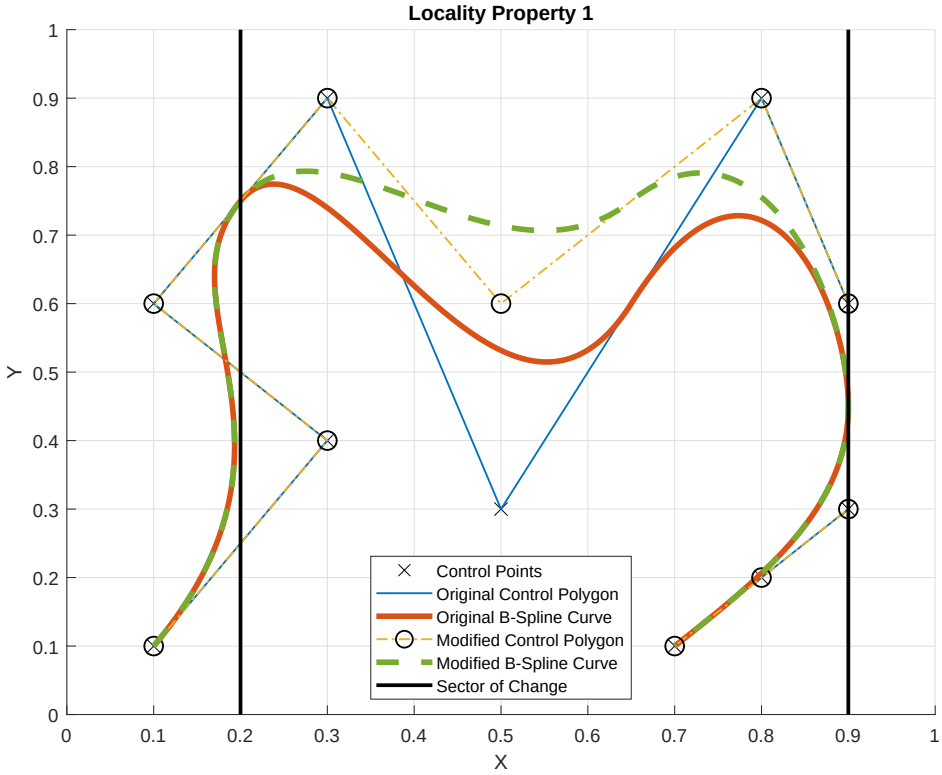
Listing 6: Proprietà di Località 2

```
% B_SPLINE_LOCALITY_2:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
```

```
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.it

clear

% Retrive inputs.
x_left_limit = 0;
x_right_limit = 1;
y_left_limit = 0;
y_right_limit = 1;
num_curve_points = 1000;
knot_vector = [0 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1 1];
degree = 3;
control_points = [0.1, 0.1; 0.3, 0.4; 0.1, 0.6; 0.3, 0.9; 0.5, 0.8; ...
                  0.8, 0.9; 0.9, 0.6; 0.9, 0.3; 0.8, 0.2; 0.7, 0.1
                      ];

% Set the figure window for drawing plots.
fig = figure('Name', 'Locality Property 2', 'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
ylabel('Y');
title('Locality Property 1');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([x_left_limit x_right_limit]);
ylim([y_left_limit y_right_limit]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(knot_vector(degree+1), knot_vector(end-degree), ...
                 num_curve_points);

% Plot control points and control polygon of the original curve.
plot(control_points(:, 1), control_points(:, 2), 'kx', 'MarkerSize', 10);
plot(control_points(:, 1), control_points(:, 2), '-', 'linewidth', 1, ...
     'color', '#0072BD');

% Calculate and plot the original B-Spline curve using De Boor algorithm.
curve = zeros(num_curve_points, 2);
```

```
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
plot(curve(:, 1), curve(:, 2), 'linewidth', 3, 'color', '#D95319');

% Choose the interval not to be change and modify the other control points.
r = 6;
control_points(1:r-degree-1, :) = control_points(1:r-degree-1, :) + 0.05;
control_points(r+1:end, :) = control_points(r+1:end, :) + 0.05;

% Plot control points and control polygon of the modified curve.
plot(control_points(:, 1), control_points(:, 2), '-.o', 'linewidth', 1, ...
     'color', '#EDB120', 'MarkerEdgeColor', 'k', 'MarkerSize', 10);

% Calculate and plot the modified B-Spline curve.
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
plot(curve(:, 1), curve(:, 2), '--', 'linewidth', 3, 'color', '#77AC30');

% Plot lines for highlighting the untouched interval.
left_line = de_boor_algorithm(control_points, degree, knot_vector, ...
                              knot_vector(r));
right_line= de_boor_algorithm(control_points, degree, knot_vector, ...
                              knot_vector(r+1));
plot([left_line(1) left_line(1)], [y_left_limit y_right_limit], 'k', ...
     'linewidth', 2)
plot([right_line(1) right_line(1)], [y_left_limit y_right_limit], 'k', ...
      'linewidth', 2)
legend({'Control Points', 'Original Control Polygon', ...
        'Original B-Spline Curve', 'Modified Control Polygon', ...
        'Modified B-Spline Curve', 'Sector of Change'}, 'Location', ...
        'south');
```

Figura 4: B-spline Località



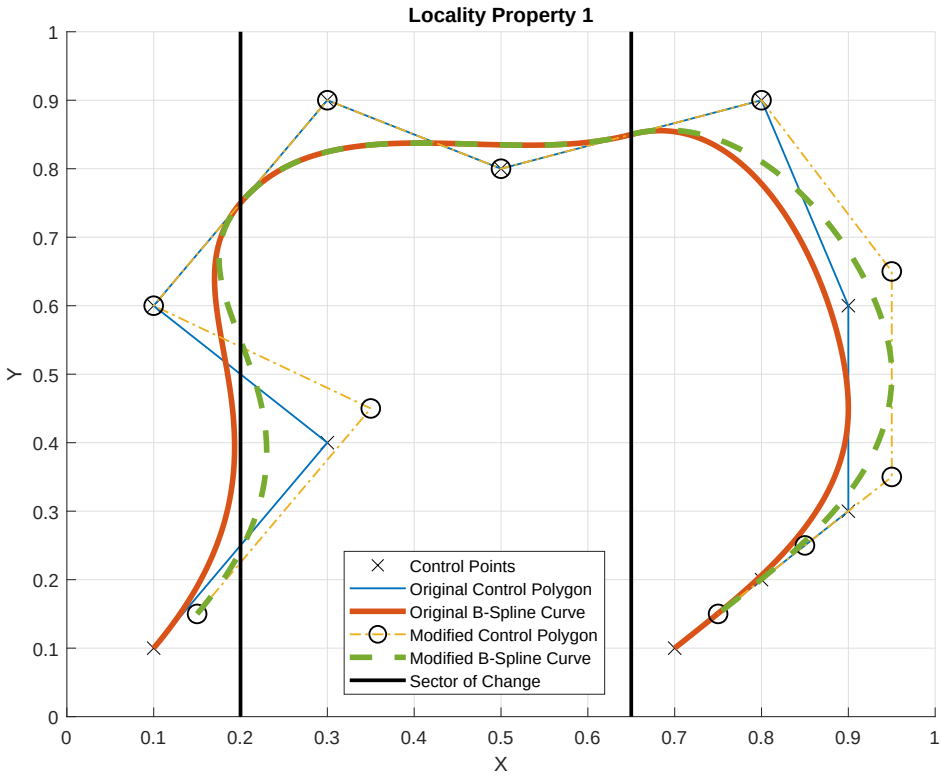
Strong Convex Hull. La curva è contenuta nel guscio convesso del suo poligono di controllo.

Variation Diminishing. Il numero di intersezioni tra la curva e una retta qualunque (un piano per le curve nello spazio) è minore o uguale al numero di intersezioni tra il poligono di controllo e tale retta (piano). Ne segue che:

- Se il poligono di controllo è convesso, la curva è convessa.
- Il numero di cambi di concavità della curva è minore o uguale al numero di cambi di concavità del poligono di controllo.

Nello script 7 è mostrata questa proprietà. Fissata una curva B-Spline, sono stati generati due punti randomici per i quali far passare una retta. Qualsiasi retta generata dallo script avrà un numero di intersezioni con la curva minore o uguale al numero di intersezioni con il poligono di controllo.

Figura 5: B-spline Località 2



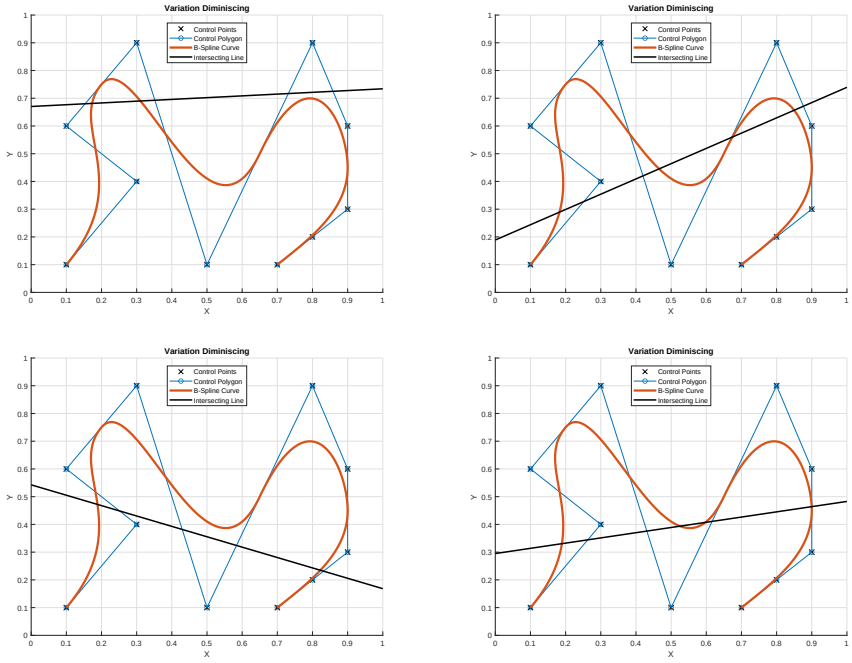


Figura 6: Variation diminishing, quattro esempi di rette

Nella figura 6 sono raffigurati quattro esempi risultanti dall'esecuzione dello script 7.

Listing 7: Variation diminishing

```
% B_SPLINE_CURVE_VARIATION_DIMINISCING:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
    it

clear

% Retrive inputs.
x_left_limit = 0;
x_right_limit = 1;
y_left_limit = 0;
y_right_limit = 1;
num_curve_points = 1000;
knot_vector = [0 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1 1];
degree = 3;
```

```
control_points = [0.1, 0.1; 0.3, 0.4; 0.1, 0.6; 0.3, 0.9; 0.5, 0.1; ...
                  0.8, 0.9; 0.9, 0.6; 0.9, 0.3; 0.8, 0.2; 0.7, 0.1];

% Set the figure window for drawing plots.
fig = figure('Name', 'Variation Diminiscing', 'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
ylabel('Y');
title('Variation Diminiscing');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([x_left_limit x_right_limit]);
ylim([y_left_limit y_right_limit]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(knot_vector(degree+1), knot_vector(end-degree), ...
                 num_curve_points);

% Plot control points and control polygon of the original curve.
plot(control_points(:, 1), control_points(:, 2), 'k.', 'MarkerSize', 20);
plot(control_points(:, 1), control_points(:, 2), '-', 'linewidth', 1, ...
     'color', '#0072BD');

% Calculate and plot the original B-Spline curve using de Boor algorithm.
curve = zeros(num_curve_points, 2);
for i = 1 : num_curve_points
    curve(i, :) = de_boor_algorithm(control_points, degree, ...
                                    knot_vector, steps(i));
end
plot(curve(:, 1), curve(:, 2), 'linewidth', 3, 'color', '#D95319');

% Generate random line to intersect with the curve.
y = 0.1 + (0.8-0.1).*rand(1, 2);
plot([0 1], y, 'k', 'linewidth', 2);
legend({'Control Points', 'Control Polygon', 'B-Spline Curve', ...
        'Intersecting Line'}, 'Location', 'best');
```

2.2 Rappresentazione di curve B-Spline chiuse

Siano $\mathbf{d}_1, \dots \mathbf{d}_m$ i vertici di controllo del poligono chiuso (con $\mathbf{d}_1 = \mathbf{d}_m$). Per definire una curva B-SPline chiusa di ordine k, si sceglie la partizione nodale

$$\mathbf{t} = \left[\frac{-k}{m-1} : \frac{1}{m-1} : \frac{k+m-1}{m-1} \right]$$

e si estende il poligono di controllo aggiungendo i vertici

$$\mathbf{d}_{m+1} = \mathbf{d}_2, \mathbf{d}_{m+2} = \mathbf{d}_3, \dots, \mathbf{d}_{m+k-1} = \mathbf{d}_k, \mathbf{d}_{m+k} = \mathbf{d}_{k+1}$$

Lo script Matlab 8, dati in input il grado della curva ed i vertici di controllo, costruisce una curva B-Spline chiusa generando la partizione nodale estesa ed estendendo il poligono di controllo come descritto sopra. In figura 7 è riportato un esempio di curva B-Spline chiusa di ordine quattro con dieci vertici di controllo generata dallo script 8 .

Listing 8: Curva B-Spline Chiusa

```
% B_SPLINE_CLOSED:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
    it

clear

% Ask user for inputs.
prompt = {'X Axis Left Limit:', 'X Axis Right Limit:', ...
          'Y Axis Left Limit:', 'Y Axis Right Limit:', ...
          ['Number of Control Points (Last one automatically' ...
          ' generated, V_1=V_n):'], 'Degree:', ...
          'Number of points of the B-Spline curves to draw:'};
inputs_title = 'Insert Inputs';
dimensions = [1 56];
default_inputs = {'0', '1', '0', '1', '5', '2', '1000'};
inputs = inputdlg(prompt, inputs_title, dimensions, default_inputs
    );
x_left_limit = str2double(inputs{1});
x_right_limit = str2double(inputs{2});
y_left_limit = str2double(inputs{3});
```

```
y_right_limit = str2double(inputs{4});
num_control_points = str2double(inputs{5});
degree = str2double(inputs{6});
num_curve_points = str2double(inputs{7});

% Get the order of the B-Spline curve.
order = degree + 1;

% Set the figure window for drawing plots.
fig = figure('Name', 'Closed B-Spline', 'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');
hold on;
grid on;
xlabel('X');
ylabel('Y');
title('Closed B-Spline');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([x_left_limit x_right_limit]);
ylim([y_left_limit y_right_limit]);

% Ask user to choose control vertices for the B-Spline curve and
    plot them.
control_points = zeros(num_control_points + degree + 2, 2);
for i = 1 : num_control_points + 1
    if i > num_control_points
        % Generate last control point V_1=V_n.
        control_points(num_control_points + 1, :) = control_points
            (1, :);
    else
        control_points(i, :) = ginput(1);
    end
    poi_plot = plot(control_points(i, 1), control_points(i, 2), 'k
        .', ...
                    'MarkerSize', 20);
    if i > 1
        pol_plot = plot(control_points(i-1:i,1), control_points(i
            -1:i, ...
                        2), '-', 'linewidth', 1, 'color', '#0072BD
                            ');
    end
end
legend([poi_plot pol_plot], {'Control Point', 'Control Polygon'},
    ...
        'Location', 'best');

% Generate knot vector.
```

```
67 knot_vector = -order/num_control_points : 1/num_control_points :
       ...
68     (order+num_control_points)/num_control_points;
69 control_points(end, :) = control_points(1, :);
70
71 % Generate last k (order) control points and plot them.
72 control_points(num_control_points+2:end, :) = control_points(2:
       degree+2,:);
73 plot(control_points(num_control_points+2: end, 1), ...
74                     control_points(num_control_points+2: end, 2),
                           'g.', ...
75                     'MarkerSize', 20, 'DisplayName', ...
76                     'New Added Control Points');
77
78 % Calculate the parameter (t) steps for drawing the B-Spline
       curves.
79 steps = linspace(knot_vector(order), knot_vector(end-degree), ...
80                  num_curve_points);
81
82 % Calculate and plot the B-Spline curve using de Boor algorithm.
83 curve = zeros(num_curve_points, 2);
84 for i = 1 : num_curve_points
85     curve(i, :) = de_boor_algorithm(control_points, degree, ...
86                                     knot_vector, steps(i));
87 end
88 plot(curve(:, 1), curve(:, 2), 'linewidth', 3, 'color', '#D95319',
       ...
89      'DisplayName', 'B-Spline Curve');
```

3 Le superfici B-Spline

3.1 Forma implicita e parametrica

Una superficie può essere rappresentata mediante equazione implicita:

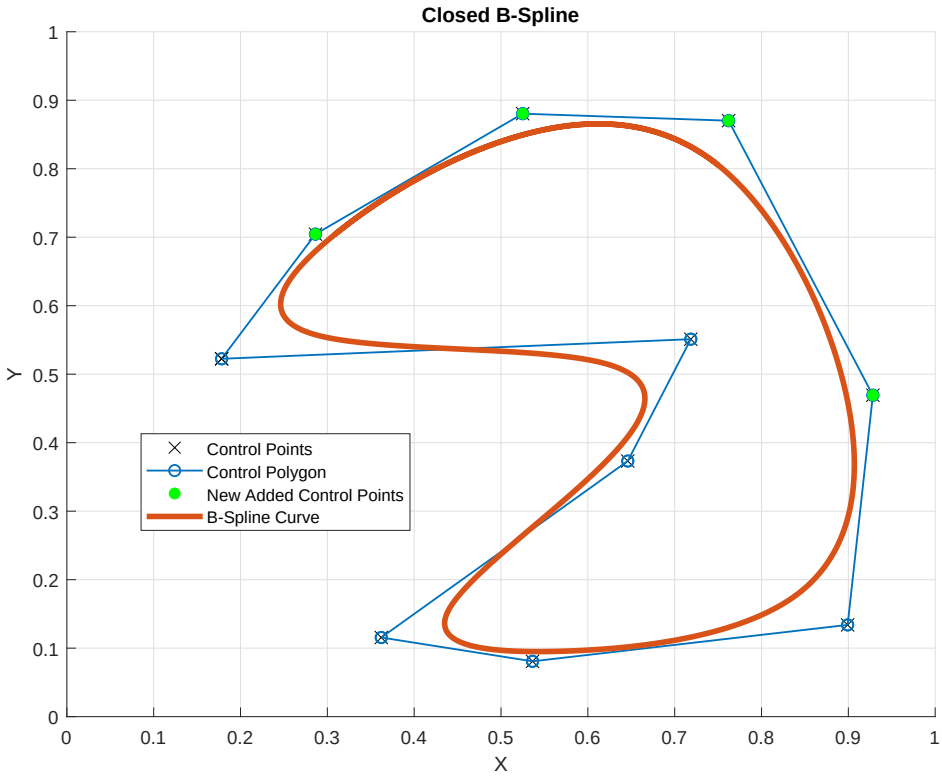
$$f(x, y, z) = 0$$

o parametrica:

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

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

Figura 7: Esempio di B-spline Chiusa



3.2 Superfici tensor-product

Siano $\{B_0^A(u), \ldots, B_n^A(u)\}$, $\{B_0^B(v), \ldots, B_m^B(v)\}$ due insiemi di funzioni linearmente indipendenti definiti sui due intervalli della retta reale

$$I_A = [a_A, b_A], I_B = [a_B, b_B]$$

con i quali si possono definire i due spazi di funzioni monovariate

$$S_1 =< B_0^A(u), \ldots, B_n^A(u) > \qquad S_2 =< B_0^B(v), \ldots, B_m^B(v) >$$

Lo spazio tensor-product $S1 \otimes S2$ dei due spazi $S1, S2$ è lo spazio delle funzioni bivariate definite sul rettangolo $R := I_A \times I_B$ che sono combinazioni lineari delle funzioni prodotto $B_i^A(u)B_i^A(v)$ al variare di i da 0 a n e j da 0 a m.

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

con $c_{i,j} \in \mathbb{R}, i = 0, \ldots, n, j = 0, \ldots, m$.
La superficie parametrica tensor-product (patch tensor-product) è definita come

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

a partire da un reticolo di (n + 1)(m + 1) punti di controllo.
Siano $U = \{u_0, \ldots, u_{n+k}\}, V = \{v_o, \ldots, v_{m+l}\}$ due vettori estesi di nodi associati agli intervalli $[a,b] = \{u_{k-1}, \ldots, u_{n+1}\}, [c,d] = \{v_{l-1}, v_{m+1}\}$, la superficie tensor-product B-Spline

$$\mathbf{X}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{d}_{i,j} N_{i,k}(u) N_{j,l}(v)$$

con $(u,v) \in [a,b] \times [c,d]$ è definita a partire da $(n+1)(m+1)$ punti di controllo di de Boor $\mathbf{d}_{i,j}, i = 0, \ldots, n, j = 0, \ldots, m$.
Nello script Matlab 9 è stato implementato il calcolo e rappresentazione di basi B-Spline di superfici dati in input i gradi delle due basi e i due vettori di nodi utilizzando la definizione di superficie tensor-product vista sopra e calcolando gli elementi delle basi mediante *Cox-de Boor* come visto precedentemente per le curve.
In figura 8 è mostrato un esempio di base B-Spline generata dallo script 9, fissati $n = m = 2$ e vettori dei nodi $t_1 = t_2 = [0,0,0,1,1,1]$.

Listing 9: Base delle superfici B-Spline

```
1 % B_SPLINE_SURFACE_BASE:
2 %
```

```
3  % Requires:
4  %   - cox_de_boor.m
5
6  % Authors: Elia Mercatanti, Marco Calamai
7  % Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
       it
8
9  clear
10
11 % Ask user for inputs.
12 prompt = {'First Base Order:', 'First Base Knocts Vector:', ...
13           'Second Base Order:', 'Second Base Knocts Vector:'};
14 inputs_title = 'Insert Inputs';
15 dimensions = [1 54];
16 default_inputs = {'3', '[0 0 0 1 1 1]', '3', '[0 0 0 1 1 1]'};
17 inputs = inputdlg(prompt, inputs_title, dimensions, default_inputs
       );
18
19 % Retrive inputs.
20 order_1 = str2double(inputs{1});
21 knot_vector_1 = str2num(inputs{2});
22 order_2 = str2double(inputs{3});
23 knot_vector_2 = str2num(inputs{4});
24 num_steps = 50;
25
26 % Initialization of the two basis matrices and steps to plot the
       surface.
27 steps_1 = linspace(knot_vector_1(order_1), knot_vector_1(end-
       order_1+1),...
28                    num_steps);
29 steps_2 = linspace(knot_vector_2(order_2), knot_vector_2(end-
       order_2+1),...
30                    num_steps);
31 num_base1_elements = length(knot_vector_1) - order_1;
32 num_base2_elements = length(knot_vector_2) - order_2;
33 first_base = zeros(num_steps, num_base1_elements);
34 second_base = zeros(num_steps, num_base2_elements);
35
36 % Calcualte the first B-Spline base.
37 for i = 1 : num_steps
38     for j = 1 : num_base1_elements
39         first_base(i, j) = cox_de_boor(j, order_1, order_1, ...
40                                        knot_vector_1, steps_1(i));
41     end
42 end
43
44 % Calcualte the second B-Spline base.
45 for i = 1 : num_steps
46     for j = 1 : num_base2_elements
```

```
47          second_base(i, j) = cox_de_boor(j, order_2, order_2, ...
48                                          knot_vector_2, steps_2(i))
                                              ;
49      end
50  end
51
52  % Set the figure window for drawing plots.
53  fig = figure('Name', 'B-Spline Surface Base', ...
54               'NumberTitle', 'off');
55  fig.Position(3:4) = [800 600];
56  movegui(fig, 'center');
57
58  % Plot the B-Spline surface.
59  for i = 1 : num_base1_elements
60      for j = 1 : num_base2_elements
61          Z = first_base(:, i)*second_base(:, j).';
62          surf(steps_1, steps_2, Z);
63          hold on;
64      end
65  end
66  grid on;
67  xlabel('X');
68  ylabel('Y');
69  zlabel('Z');
70  title('B-Spline Surface Base');
```

Nello script 10 e relativa immagine 9 è stata rappresentata una curva B-Spline utilizzando la definizione di superficie tensor-product vista in precedenza. Sono state inoltre rappresentate le curve di bordo della superficie. I bordi di una superficie B-Spline sono definiti come:

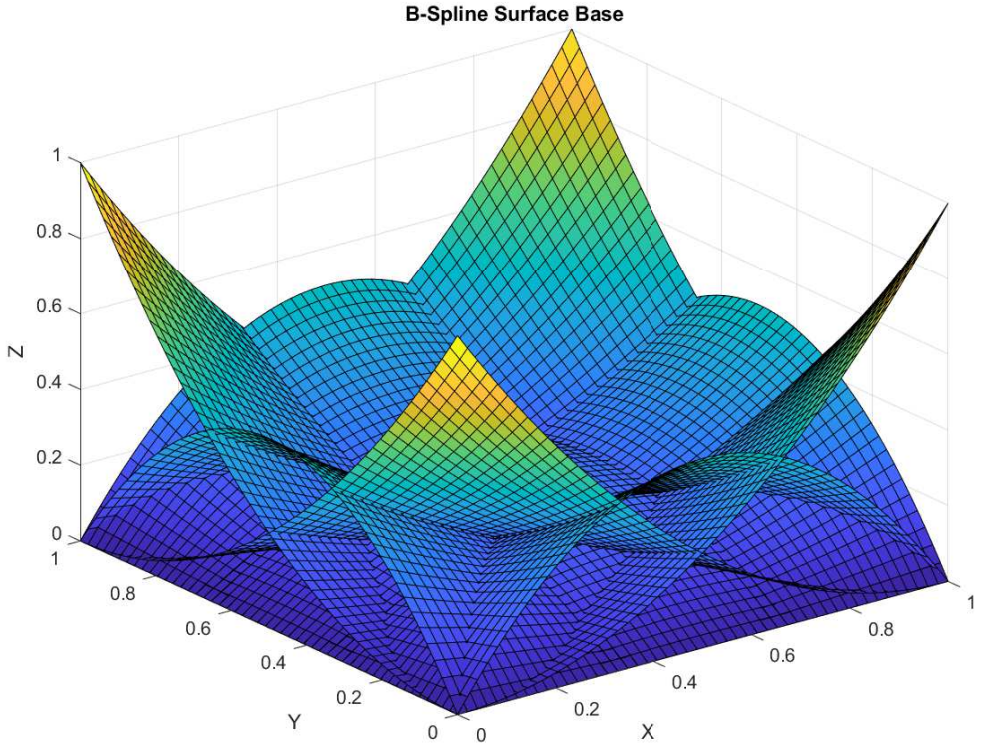
$$\mathbf{X}(a,v) = \sum_{j=0}^{m} \mathbf{d}_{0,j} N_{j,l}(v) \quad \mathbf{X}(b,v) = \sum_{j=0}^{m} \mathbf{d}_{n,j} N_{j,l}(v)$$

$$\mathbf{X}(u,c) = \sum_{i=0}^{n} \mathbf{d}_{i,0} N_{i,k}(u) \quad \mathbf{X}(u,d) = \sum_{i=0}^{n} \mathbf{d}_{i,m} N_{i,k}(u)$$

Listing 10: Superficie B-Spline

```
1  % B_SPLINE_SURFACE_TENSOR_PRODUCT:
2  %
3  % Requires:
4  %   - cox_de_boor.m
5
6  % Authors: Elia Mercatanti, Marco Calamai
7  % Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
       it
```

Figura 8: Base superficie B-Spline




```
8
9   clear
10
11  % Retrive inputs.
12  control_grid_x = [4.5 3.5 2.5; 4.5 3.5 2.5; 4.5 3.5 2.5];
13  control_grid_y = [4.5 4.5 4.5; 3.5 3.5 3.5; 1.5 1.5 1.5];
14  control_grid_z = [0 0 0; 2.6 2.6 2.6; 0 0 0];
15  order_1 = 3;
16  order_2 = 3;
17  knot_vector_1 = [0 0 0 1 1 1];
18  knot_vector_2 = [0 0 0 1 1 1];
19  num_steps = 50;
20
21  % Initialization of the two basis matrices and steps to plot the
        surface.
22  steps_1 = linspace(knot_vector_1(order_1), knot_vector_1(end-
        order_1+1),...
23                     num_steps);
```

```
steps_2 = linspace(knot_vector_2(order_2), knot_vector_2(end-
    order_2+1),...
                   num_steps);
num_base1_elements = length(knot_vector_1) - order_1;
num_base2_elements = length(knot_vector_2) - order_2;
first_base = zeros(num_steps, num_base1_elements);
second_base = zeros(num_steps, num_base2_elements);

% Calcualte the first B-Spline base.
for i = 1 : num_steps
    for j = 1 : num_base1_elements
        first_base(i, j) = cox_de_boor(j, order_1, order_1, ...
                                       knot_vector_1, steps_1(i));
    end
end

% Calcualte the second B-Spline base.
for i = 1 : num_steps
    for j = 1 : num_base2_elements
        second_base(i, j) = cox_de_boor(j, order_2, order_2, ...
                                        knot_vector_2, steps_2(i))
                                            ;
    end
end

% Set the figure window for drawing plots.
fig = figure('Name', 'B-Spline Surface with Tensor Product', ...
             'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');

% Calculate tensor product and plot the B-Spline surface.
surf_x = first_base*control_grid_x*second_base.';
surf_y = first_base*control_grid_y*second_base.';
surf_z = first_base*control_grid_z*second_base.';
surf(surf_x , surf_y , surf_z);
hold on;
grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('B-Spline Surface with Tensor Product and Edge Curves');
axes = gca;

% Plot the control grid of the B-Spline surface.
pol_plot = plot3(control_grid_x, control_grid_y, control_grid_z,
    ...
                 'b.--', 'linewidth', 2, 'MarkerSize', 25);
plot3(control_grid_x', control_grid_y', control_grid_z', 'b--',
```

```
    ...
70      'linewidth', 2);
71
72  % Plot edge curves and legend.
73  set(axes, 'ColorOrder', circshift(get(gca, 'ColorOrder'), -1))
74  edge_curve1_plot = plot3(surf_x(1, :), surf_y(1, :), surf_z(1, :),
        ...
75                           'linewidth', 10);
76  edge_curve2_plot = plot3(surf_x(:, 1), surf_y(:, 1), surf_z(:, 1),
        ...
77                           'linewidth', 10);
78  edge_curve3_plot = plot3(surf_x(end, :), surf_y(end, :), ...
79                           surf_z(end, :), 'linewidth', 10);
80  edge_curve4_plot = plot3(surf_x(:, end), surf_y(:, end), ...
81                           surf_z(:, end), 'linewidth', 10);
82  axis tight;
83  axis equal;
84  legend([pol_plot(1) edge_curve1_plot edge_curve2_plot
        edge_curve3_plot ...
85          edge_curve4_plot], {'Control Grid', 'First Edge Curve',
              ...
86          'Second Edge Curve', 'Third Edge Curve', 'Fourth Edge
              Curve'}, ...
87          'Location', 'best');
```

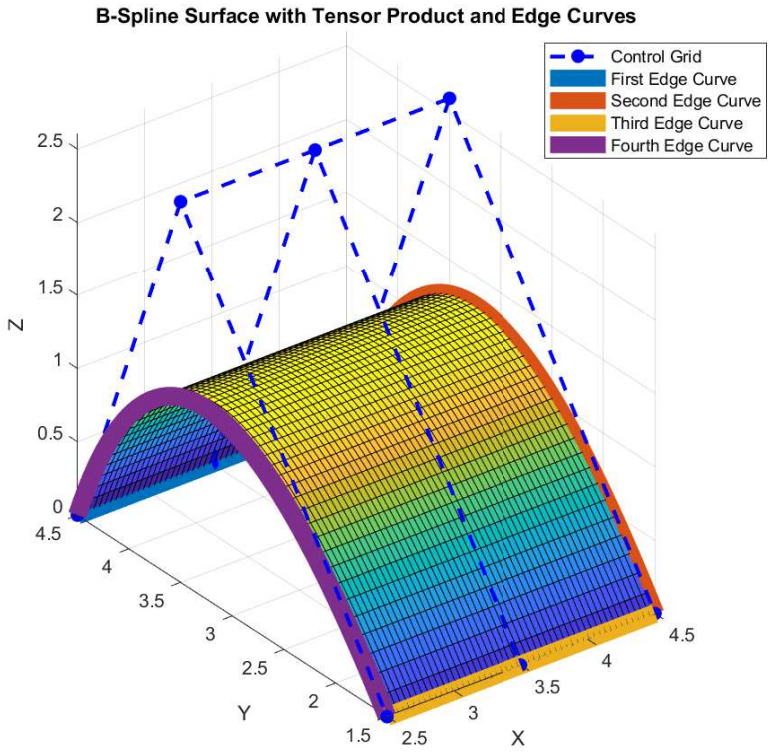
3.3 Proprietà di invarianza per trasformazioni affini

Anche le superfici B-SPline godono della proprietà di invarianza per trasformazioni affini vista in precedenza per le curve. L'importanza pratica di questa proprietà è la seguente: supponiamo di aver disegnato una superficie e di volerle applicare una certa trasformazione affine (rotazione, traslazione, scala, ...). Il modo più semplice di procedere è applicare ai vertici di controllo la trasformazione affine desiderata e poi ridisegnare la curva. Nello script Matlab 11 è stata inizialmente definita e disegnata una superficie B-Spline e successivamente applicata una trasformazione affine. In particolare, sono state applicate in quest'ordine, una rotazione, traslazione e scalatura inizialmente ai vertici di controllo e successivamente direttamente sulla curva originale, ottenendo la stessa B-spline come mostrato in figura 10 . La proprietà di invarianza per trasformazioni affini viene confermata dal fatto che le due superfici trasformate combaciano.

Listing 11: Trasformazione affine superficie B-Spline

```
1  % B_SPLINE_SURFACE_TRANS:
2  %
3  % Requires:
4  %   - cox_de_boor.m
```

Figura 9: Superficie B-Spline e relative curve di bordo




```
5
6  % Authors: Elia Mercatanti, Marco Calamai
7  % Emails: elia.mercatanti@stud.unifi.it, marco.calamai@stud.unifi.
       it
8
9  clear
10
11 % Retrive inputs.
12 control_grid_x = [4.5 3.5 2.5; 4.5 3.5 2.5; 4.5 3.5 2.5];
13 control_grid_y = [4.5 4.5 4.5; 3.5 3.5 3.5; 1.5 1.5 1.5];
14 control_grid_z = [0 0 0; 2.6 2.6 2.6; 0 0 0];
15 order_1 = 3;
16 order_2 = 3;
17 knot_vector_1 = [0 0 0 1 1 1];
18 knot_vector_2 = [0 0 0 1 1 1];
19 num_steps = 50;
20
21 % Initialization of the two basis matrices and steps to plot the
```

```
    surface.
steps_1 = linspace(knot_vector_1(order_1), knot_vector_1(end-order_1+1),...
                   num_steps);
steps_2 = linspace(knot_vector_2(order_2), knot_vector_2(end-order_2+1),...
                   num_steps);
num_base1_elements = length(knot_vector_1) - order_1;
num_base2_elements = length(knot_vector_2) - order_2;
first_base = zeros(num_steps, num_base1_elements);
second_base = zeros(num_steps, num_base2_elements);

% Calcualte the first B-Spline base.
for i = 1 : num_steps
    for j = 1 : num_base1_elements
        first_base(i, j) = cox_de_boor(j, order_1, order_1, ...
                                       knot_vector_1, steps_1(i));
    end
end

% Calcualte the second B-Spline base.
for i = 1 : num_steps
    for j = 1 : num_base2_elements
        second_base(i, j) = cox_de_boor(j, order_2, order_2, ...
                                        knot_vector_2, steps_2(i))
                                            ;
    end
end

% Set the figure window for drawing plots.
fig = figure('Name', 'Affine Transformations on B-Spline Surface', ...
             'NumberTitle', 'off');
fig.Position(3:4) = [800 600];
movegui(fig, 'center');

% Plot the original B-Spline surface.
surf_x = first_base*control_grid_x*second_base.';
surf_y = first_base*control_grid_y*second_base.';
surf_z = first_base*control_grid_z*second_base.';
origin_surf_plot = surf(surf_x , surf_y , surf_z, 'FaceColor', 'r');
hold on;
grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Affine Transformations on B-Spline Surfaces');

```

```
% Plot the original control grid of the B-Spline surface.
origin_pol_plot = plot3(control_grid_x, control_grid_y,
    control_grid_z, ...
                        'b.--', 'linewidth', 2, 'MarkerSize', 25);
plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'b--',
    ...
      'linewidth', 2);

% Tranlation, rotation and scaling transformations.
translation = [4 1 1];
rotation = [cos(pi/2) -sin(pi/2) 0; sin(pi/2) cos(pi/2) 0; 0 0 1];
scaling = [0.8 0 0; 0 0.8 0; 0 0 0.8];

% Transform control points into a single matrix for applying
% transformations (num_control_points x dimensions).
control_points = [reshape(control_grid_x.', [], 1), ...
                  reshape(control_grid_y.', [], 1), ...
                  reshape(control_grid_z.', [], 1)];

% Transformation on control points.
control_points = (control_points*rotation + translation)*scaling;

% Restore control points in three matrices (control grid).
control_grid_x = reshape(control_points(:, 1), order_1, order_2)
    .';
control_grid_y = reshape(control_points(:, 2), order_1, order_2)
    .';
control_grid_z = reshape(control_points(:, 3), order_1, order_2)
    .';

% Plot the transformed B-Spline surface.
surf_x_trans = first_base*control_grid_x*second_base.';
surf_y_trans = first_base*control_grid_y*second_base.';
surf_z_trans = first_base*control_grid_z*second_base.';
trasf_control_plot = surf(surf_x_trans, surf_y_trans, surf_z_trans
    , ...
                          'FaceColor', 'b');

% Plot the control grid of the B-Spline surface.
trasf_pol_plot = plot3(control_grid_x, control_grid_y,
    control_grid_z, ...
                       'r.-', 'linewidth', 2, 'MarkerSize', 25);
plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'r-',
    ...
      'linewidth', 2);

% Transform surface points matrices into a single matrix for
    applying
% transformations (num_control_points x dimensions).
```

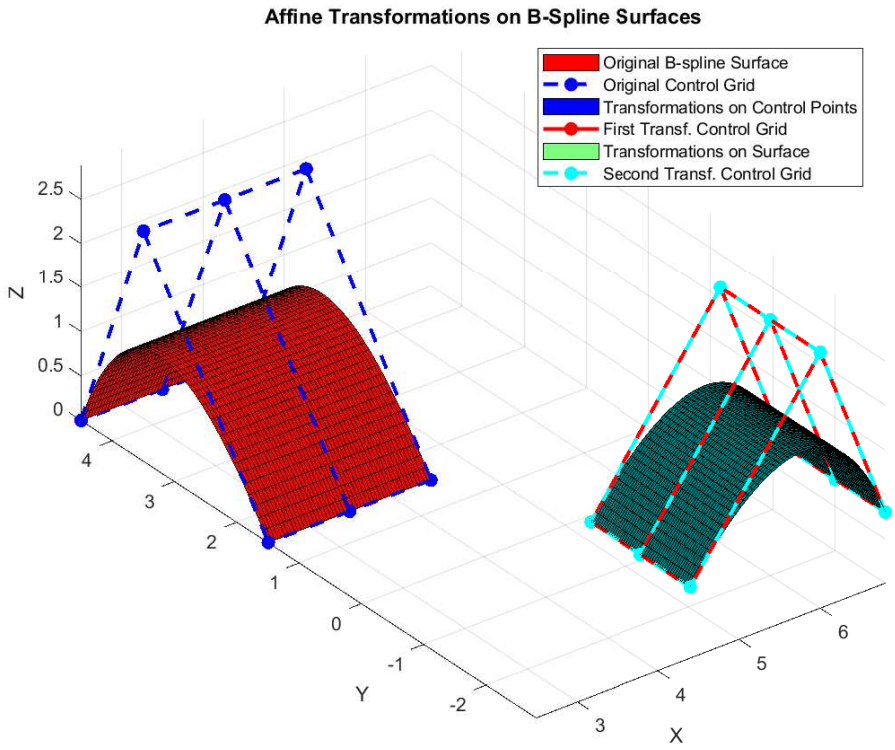
```
105 surface_points = [reshape(surf_x.', [], 1), reshape(surf_y.', [],
    1), ...
106                   reshape(surf_z.', [], 1)];
107
108 % Transformation on surface points.
109 surface_points = (surface_points*rotation + translation)*scaling;
110
111 % Restore surface points in three matrices (surf).
112 surf_x = reshape(surface_points(:, 1), num_steps, num_steps).';
113 surf_y = reshape(surface_points(:, 2), num_steps, num_steps).';
114 surf_z = reshape(surface_points(:, 3), num_steps, num_steps).';
115
116 % Plot the transformed B-Spline surface.
117 trasf_surf_plot = surf(surf_x , surf_y , surf_z, 'FaceColor', 'g',
    ...
118                        'FaceAlpha', 0.5);
119
120 % Plot the control grid of the B-Spline surface and legend.
121 trasf_surf_pol_plot = plot3(control_grid_x, control_grid_y, ...
122                             control_grid_z, 'c.--', 'linewidth',
                                2, ...
123                             'MarkerSize', 25);
124 plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'c--',
    ...
125       'linewidth', 2);
126 axis tight;
127 axis equal;
128 legend([origin_surf_plot origin_pol_plot(1) trasf_control_plot ...
129         trasf_pol_plot(1) trasf_surf_plot ...
130         trasf_surf_pol_plot(1)], {'Original B-spline Surface', ...
131         'Original Control Grid', 'Transformations on Control
              Points',...
132         'First Transf. Control Grid', 'Transformations on Surface'
              , ...
133         'Second Transf. Control Grid'}, 'Location', 'best');
```

3.4 Algoritmo di de Boor

L'algoritmo di de-Boor può essere esteso per calcolare anche le superfici B-SPline. Più precisamente, l'algoritmo può essere applicato diverse volte, fino a che non si ottiene il punto corrispondente sulla superficie B-Spline $p(u, v)$ dato (u, v). Dato quindi:

$$p(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u) N_{j,l}(v) \mathbf{d}_{i,j}$$

Figura 10: Risultato di una trasformazione affine applicata ad una superficie B-Spline



Invece di calcolare la superficie effettuando le operazioni in cascata, possiamo porre:

$$q_i(v) = \sum_{j=0}^{m} N_{j,l}(v)\mathbf{d}_{i,j}$$

Come si può notare, $q_i(v)$ è un punto sulla curva B-Spline definita dai punti di controllo $p_{i,0}, p_{i,1}, \ldots, p_{i,m}$. A questo punto, si può utilizzare l'algoritmo di de Boor per calcolare q_i per ogni i.
Si ottiene quindi

$$P(u,v) = \sum_{i=0}^{n} N_{i,k}(u)q_i(v)$$

Possiamo quindi utilizzare nuovamente utilizzare l'algoritmo di de Boor.
Quindi, riassumendo, quello che si fa è applicare m volte l'algoritmo di de Boor

per calcolare $q_i(v)$ per ogni i e poi altre n volte per calcolare $p(u, v)$ per ogni j. Lo script 12 implementa l'algoritmo di de Boor modificato per calcolare le superfici B-Spline come spiegato sopra, mentre nello script 13 è stata rappresentata mediante l'algoritmo di de Boor la superficie B-Spline vista in precedenza. Il risultato di quest'ultimo è mostrato in figura 11.

Listing 12: Algoritmo di de Boor per le superfici B-Spline

```
function [surface_point] = de_boor_algorithm_surface(
   control_points, ...
        order_1, order_2, knot_vector_1, knot_vector_2, t_1_star,
           t_2_star)

    % Uses order_1 times de Boor algorithm to calculate order_1
       points.
    Q = zeros(order_1, 3);
    for i = 1 : order_1
        Q(i, :) =  de_boor_algorithm(control_points(order_2*(i-1)
           +1: ...
                                    order_2*i, :), order_2-1, ...
                                    knot_vector_2, t_2_star);
    end

    % Uses de Boor algorithm on Q to calculate the final surface
       point.
    surface_point = de_boor_algorithm(Q, order_1-1, knot_vector_1,
        ...
                                     t_1_star);
end
```

Listing 13: Rappresentazione superficie B-Spline mediante algoritmo di de Boor

```
% B_SPLINE_SURFACE_DE_BOOR:
%
% Requires:
%   - de_boor_algorithm.m

% Authors: Elia Mercatanti, Marco Calamai
% Emails: elia.mercatanti@stud.unifi.it, marco.calamai@
   stud.unifi.it

clear

% Retrive inputs.
control_points = [4.5 4.5 0; 3.5 4.5 0; 2.5 4.5 0; 4.5
   3.5 2.6; ...
```

```
13                  3.5 3.5 2.6; 2.5 3.5 2.6; 4.5 1.5 0;
                       3.5 1.5 0; ...
14                  2.5 1.5 0];
15 order_1 = 3;
16 order_2 = 3;
17 knot_vector_1 = [0 0 0 1 1 1];
18 knot_vector_2 = [0 0 0 1 1 1];
19 num_steps = 50;
20
21 % Initialization of the two basis matrices and steps to
      plot the surface.
22 steps_1 = linspace(knot_vector_1(order_1), knot_vector_1
      (end-order_1+1),...
23                  num_steps);
24 steps_2 = linspace(knot_vector_2(order_2), knot_vector_2
      (end-order_2+1),...
25                  num_steps);
26
27 % Calculate with de boor every point of the B-Spline
      surface.
28 surface_points = zeros(num_steps*num_steps, 3);
29 counter = 1;
30 for i = 1 : num_steps
31     for j = 1 : num_steps
32         surface_points(counter, :) =
              de_boor_algorithm_surface( ...
33             control_points, order_1, order_2,
                  knot_vector_1, ...
34             knot_vector_2, steps_1(i), steps_2(j));
35         counter = counter + 1;
36     end
37 end
38
39 % Set the figure window for drawing plots.
40 fig = figure('Name', 'B-Spline Surface with De Boor',
      ...
41              'NumberTitle', 'off');
42 fig.Position(3:4) = [800 600];
43 movegui(fig, 'center');
44
```

```
45 % Plot the b-spline surface.
46 surface_matrix_x = reshape(surface_points(:, 1),
      num_steps, num_steps).';
47 surface_matrix_y = reshape(surface_points(:, 2),
      num_steps, num_steps).';
48 surface_matrix_z = reshape(surface_points(:, 3),
      num_steps, num_steps).';
49 surf(surface_matrix_x, surface_matrix_y,
      surface_matrix_z);
50 hold on;
51 grid on;
52 xlabel('X');
53 ylabel('Y');
54 zlabel('Z');
55 title('B-Spline Surface with De Boor');
56
57 % Puts control points in three matrices (control grid).
58 control_grid_x = reshape(control_points(:, 1), order_1,
      order_2).';
59 control_grid_y = reshape(control_points(:, 2), order_1,
      order_2).';
60 control_grid_z = reshape(control_points(:, 3), order_1,
      order_2).';
61
62 % Plot the control grid of the B-Spline surface.
63 pol_plot = plot3(control_grid_x, control_grid_y,
      control_grid_z, ...
64                  'b.--', 'linewidth', 2, 'MarkerSize',
                       25);
65 plot3(control_grid_x.', control_grid_y.', control_grid_z
      .', 'b--', ...
66       'linewidth', 2);
67 axis tight;
68 axis equal;
69 legend(pol_plot(1), {'Control Grid'}, 'Location', 'best'
      );
```

Figura 11: Rappresentazione superficie B-Spline mediante algoritmo di de Boor