Numerical Methods For Graphics

Elaborato

Lorenzo Pratesi Mariti

Curve e Superfici B-spline

Università degli Studi di Firenze
A.A. 2020-2021

Indice

1 Base delle B-spline

Definiamo una base che permette di scrivere algoritmi efficienti per le spline. La base è definita a partire da un *vettore esteso dei nodi*

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

con

$$\mathbf{t}_0 \leq t_1 \leq \cdots \leq t_{k-2} \leq \underbrace{t_{k-1}}_{\tau_0} < t_k < \cdots < t_n < \underbrace{t_{n+1}}_{\tau_L} \leq t_{n+2} \leq \cdots \leq t_{n+k}$$

dove la parte definita da $t_{k-1}, \ldots, t_{n+1}$ corrisponde con i nodi di τ. In pratica si aggiungono $k-1$ nodi a destra e a sinistra, questi sono detti *nodi ausiliari* e sono tanti quanto è il grado. Con un k generico, si ottiene per differenza di indici che $L+1 = n+1-k+2$ e $n = k+L-2$. La sua dimensione è $k+(L-1)$ che, espressa in termini di n, si ottiene $\dim S_{m,\tau} = n+1$.

La definizione delle B-spline è ricorsiva. Definiamo quello di ordine $k=1$, con continuità C^0,

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

Le B-spline di ordine $r \leq k$ sono definite ricorsivamente (per $r = 2, \ldots, k$) come

$$N_{i,r}(t) = \frac{t - t_i}{t_{i+r-1} - t_i} N_{i,r-1}(t) + \frac{t_{i+r} - t}{t_{i+r} - t_{i+1}} N_{i+1,r-1}(t)$$

oppure mediante la funzione $\omega_{i,r}(t)$ come

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

per $i = 0, \ldots, n+k-r$, dove

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

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

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

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

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

La definizione della base delle B-spline di *Cox - De Boor* non cambia, si usa la stessa definizione ricorsiva, ma $\omega_{i,r}(t)$ può essere identicamente nullo per qualche r a causa di nodi multipli, ciò implica una riduzione di regolarità per le B-spline il cui insieme di nodi attivi (ovvero quelli non nulli) contiene un nodo multiplo: $C^{k-m_i-1}(\tau_i)$, ovvero $C^{m-m_i}(\tau_i)$. Ogni volta che aumentiamo la molteplicità di un nodo andiamo a diminuire la continuità nello spazio considerato.

Nel seguente codice Matlab sono state implementate le funzioni descritte sopra per il calcolo delle basi di *Cox - De Boor*. La funzione principale è `bspline_basis_recurrence`, la quale si occupa di calcolare le basi delle B-spline richiamando ricorsivamente se stessa.
A livello implementativo, nella funzione `bspline_basis_recurrence` 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 il calcolo di $\omega_{ir}(t)$, viene effettuato se e solo se il denominatore e non nullo.

```
function [y, x] = bspline_basis(i, k, t, x)

if nargin < 4
    x = linspace(t(1), t(end), 1000);  % allocate points uniformly

y = bspline_basis_recurrence(i, k, t, x);

function y = bspline_basis_recurrence(i, r, t, t_star)

y = zeros(size(t_star));
if r > 1
    N_1 = bspline_basis(i, r-1, t, t_star);
    omega1_num = t_star - t(i+1);
    omega1_den = t(i+r) - t(i+1);

    if omega1_den ~= 0  % indeterminate forms 0/0 are deemed to be zero
        y = y + N_1.*(omega1_num./omega1_den);
    end

    N_2 = bspline_basis(i+1, r-1, t, t_star);
```

```
    omega2_num = t(i+r+1) - t_star;
    omega2_den = t(i+r+1) - t(i+1+1);

    if omega2_den ~= 0
        y = y + N_2.*(omega2_num./omega2_den);
    end

elseif t(i+2) < t(end)  % treat last element of knot vector as a special case
    y(t(i+1) <= t_star & t_star < t(i+2)) = 1;
else
    y(t(i+1) <= t_star) = 1;
end
```

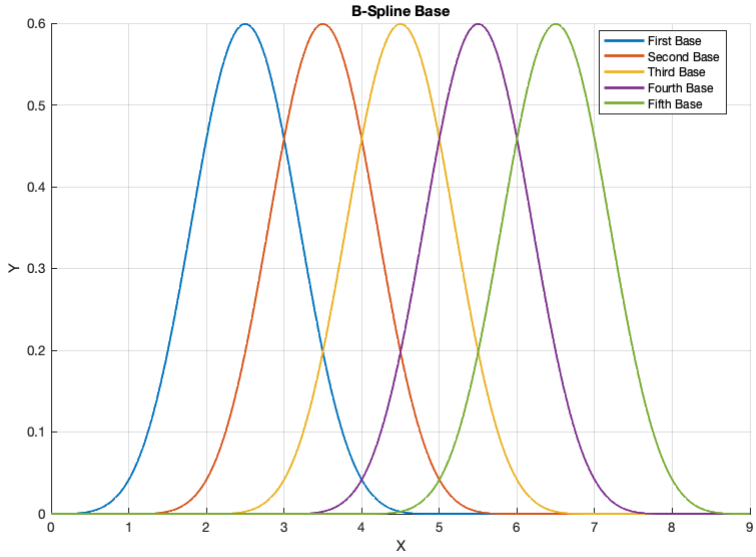
Per disegnare degli esempi di basi B-spline viene utilizzato lo sript sottostante in cui si richiama la funzione `bspline_basis` per il calcolo delle basi di *Cox - de Boor*. Vediamo degli esempi.

```
% Illustrates B-spline basis functions.
t = [0 0 0 0 0 0 1 1 1 1 1 1];  % knot vector
k = 6; % degree of the spline basis

title('B-spline Base');
xlabel('X');ylabel('Y');
hold all; grid on;

% Calculate and plot curves for the elements of the base using
% Cox-de Boor recursion formula.
for j = 0 : numel(t) - k - 1
    [y, x] = bspline_basis(j, k, t);

    ordinal = iptnum2ordinal(j+1);
    plot(x, y, 'linewidth', 2, 'DisplayName', ...
        [upper(ordinal(1)), ordinal(2:min(end)) ' Base']);
end

legend('Location', 'best');
```

1.1 Esempi di basi

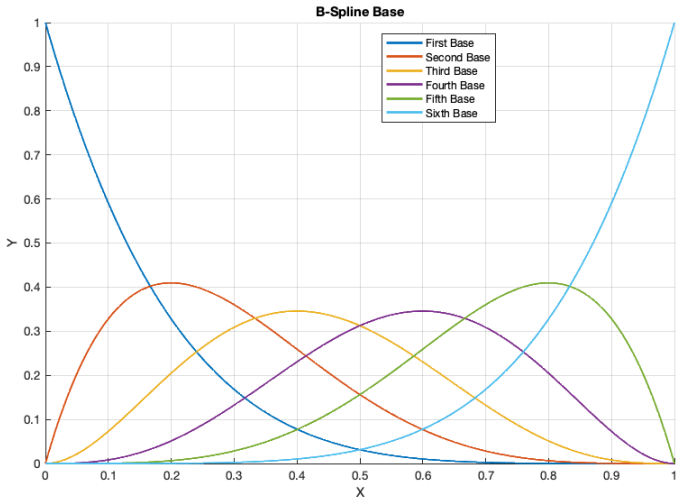
L'esempio più immediato di base è quello dove il vettore esteso dei nodi è uniforme, in questo caso abbiamo preso $\mathbf{t} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ e $k = 5$, otterremo quindi 5 funzioni di base visualizzate in Figura 1.

Figura 1: Esempio di base con $t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ e $k = 5$



Cambiando il vettore esteso dei nodi $\mathbf{t}$ otteniamo basi per le B-spline con regolarità diversa. Un caso particolare della base delle B-spline sono i polinomi di *Bernstein*. Questi ultimi si ottengono quando, dato $[a, b] = [\tau_0, \tau_L]$, la partizione nodale estesa è formata solamente da a ripetuto k volte e b ripetuto altrettante k volte. In Figura 2 è mostrato un esempio di base ottenuta con i polinomi di *Besrnstein* di grado 6.

Figura 2: Esempio di base con $t = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]$ e $k = 6$



2 Le Curve B-spline

Dati $n + 1$ punti di controllo, una curva B-spline

$$\mathbf{x} : [a, b] = [\tau_0, \tau_L] \to \mathbb{E}^d, \quad \text{con} \quad \tau_0 < \tau_1 < \cdots < \tau_{L-1} < \tau_L$$

è una curva parametrica della forma

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{d}_i N_{i,k}(t), \qquad t \in [a, b] = [\tau_0, \tau_L] = [t_{k-1}, t_{n+1}],$$

le cui componenti sono funzioni spline di grado m (ordine $k = m + 1$) espresse nella base delle B-spline sul vettore esteso dei nodi $\mathbf{t}$, dove ogni τ_i è ripetuto m_i volte, $i = 1, \ldots, L-1$. I $\mathbf{d}_i = d_0, \ldots, d_n$ sono i punti di controllo, detti di *de Boor*.

2.1 Algoritmo di de Boor

Le curve B-spline presenti in questa relazione sono state calcolate e rappresentate utilizzando l'algoritmo di De Boor.
Si tratta di un analogo del *de Casteljeau*, è sempre recursivo. Si usa per calcolare le B-spline parametriche sul calcolatore. 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 $degree - m + 1$ 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 $degree - degree + 1 = 1$) 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 t 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 t.
Nella funzione Matlab 1 è stato implementato l'algoritmo di De Boor per il calcolo e la rappresentazione di curve B-spline. Supponendo che il nodo t si trovi nel knot span $[t_l, t_{l+1})$, vengono copiati i punti di controllo che saranno influenzati dall'algoritmo $\mathbf{d}_{l-s}, \mathbf{d}_{l-s-1}, \mathbf{d}_{l-s-2}, \ldots, \mathbf{d}_{l-degree+1}, \mathbf{d}_{l-degree}$ in un nuovo array e rinominati come: $\mathbf{d}_{l-s,0}, \mathbf{d}_{l-s-1,0}, \mathbf{d}_{l-s-2,0}, ..., \mathbf{d}_{l-degree+1,0}, \mathbf{d}_{l-degree,0}$ dove lo 0 indica lo step iniziale (che ovviamente crescerà ad ogni passo dell'algoritmo).

Listing 1: Algoritmo di de Boor

```
function [C,t_star] = bspline_deboor(order,knot_vector,control_points,t_star)
% Evaluate explicit B-spline at specified locations.
%
% Input arguments:
% order:
%    B-spline order (2 for linear, 3 for quadratic, etc.)
% knot_vector:
%    knot vector
% control_points:
%    control points, typically 2-by-m, 3-by-m or 4-by-m (for weights)
% t_star (optional):
%    values where the B-spline is to be evaluated, or a positive
%    integer to set the number of points to automatically allocate
%
% Output arguments:
% C:
%    points of the B-spline curve

% B-spline polynomial degree (1 for linear, 2 for quadratic, etc.)
d = order-1;

% allocate points uniformly
if nargin < 4
    t_star = linspace(knot_vector(d+1), knot_vector(end-d), ...
                10*size(control_points,2));
elseif isscalar(t_star) && t_star > 1
```

```
    t_star = linspace(knot_vector(d+1), knot_vector(end-d), t_star);
end

m = size(control_points,1);  % dimension of control points
knot_vector = knot_vector(:).';     % knot sequence
t_star = t_star(:);

% Calculate multiplicity of t_star in the knot vector (0<=s<=degree+1)
S = sum(bsxfun(@eq, t_star, knot_vector), 2);

% Find index of knot interval that contains t_star.
I = bspline_deboor_interval(t_star,knot_vector);

P_ir = zeros(m,d+1,d+1);
a_ir = zeros(d+1,d+1);

C = zeros(size(control_points,1), numel(t_star));
for j = 1 : numel(t_star)
    u = t_star(j);
    s = S(j);
    ix = I(j);
    P_ir(:) = 0;
    a_ir(:) = 0;

    % identify d+1 relevant control points
    P_ir(:, (ix-d):(ix-s), 1) = control_points(:, (ix-d):(ix-s));
    h = d - s;

    if h > 0
        % de Boor recursion formula
        for r = 1 : h
            q = ix-1;
            for i = (q-d+r) : (q-s)
                a_ir(i+1,r+1) = (u-knot_vector(i+1)) / ...
                                (knot_vector(i+d-r+1+1)-knot_vector(i+1));

                P_ir(:,i+1,r+1) = (1-a_ir(i+1,r+1)) * P_ir(:,i,r) + ...
                                  a_ir(i+1,r+1)    * P_ir(:,i+1,r);
            end
        end
```

```
67
68          % extract value from triangular computation scheme
69          C(:,j) = P_ir(:,ix-s,d-s+1);
70      elseif ix == numel(knot_vector)  % last control point is a special case
71          C(:,j) = control_points(:,end);
72      else
73          C(:,j) = control_points(:,ix-d);
74      end
75  end
76
77  function ix = bspline_deboor_interval(t_star,knot_vector)
78  % Index of knot in knot sequence not less than the value of t_star.
79  % If knot has multiplicity greater than 1, the highest index is returned.
80
81  % indicator of knot interval in which t_star is
82  i = bsxfun(@ge, t_star, knot_vector) & ...
83      bsxfun(@lt, t_star, [knot_vector(2:end) 2*knot_vector(end)]);
84  [row,col] = find(i);
85  [row,ind] = sort(row);  % restore original order of data points
86  ix = col(ind);
```

2.2 Proprietà delle curve B-spline

Le curve B-spline godono di diverse proprietà:

1. Invarianza per trasformazioni affini: la proprietà della base delle B-spline di essere una partizione dell'unità garantisce che le curve B-spline siano invarianti per trasformazioni affini. Questo vuol dire che l'applicazione della trasformazione affine sulla curva, o sui punti di controllo, è indifferente in quanto il risultato non cambia.

2. Località: un segmento di curva è influenzato solamente da k punti di controllo.

3. Strong Convex Hull: ogni punto sulla curva appartiene all'inviluppo convesso di k punti di controllo consecutivi, con k ordine delle funzioni spline.

4. Variation Diminishing: il numero di intersezioni tra una retta e la curva è minore o uguale al numero di intersezioni tra la stessa retta ed il poligono di controllo.

2.2.1 Invarianza per trasformazioni affini

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.

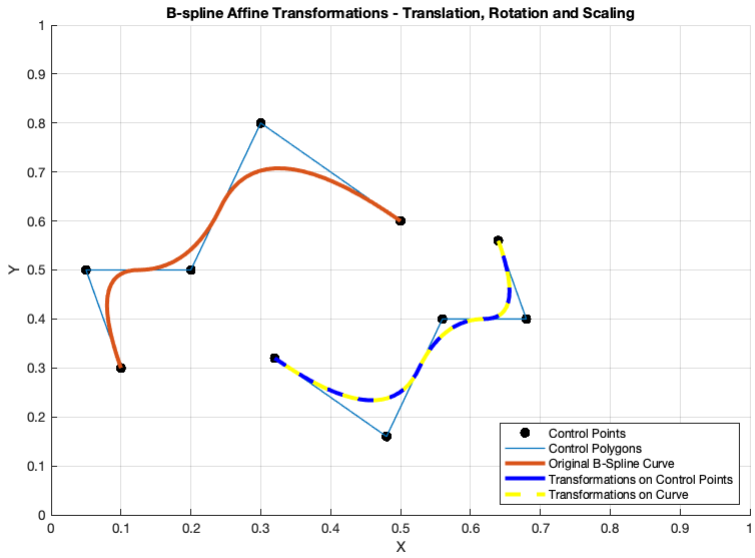
Nello Script Matlab 2 è stata inizialmente definita e disegnata una curva B-spline e successivamente applicata una trasformazione affine prima ai suoi punti di controllo e successivamente alla curva. In particolare sono state applicate in quest'ordine: rotazione, traslazione e scalatura, inizialmente ai vertici di controllo originali, ottenendo la curva di colore arancione mostrata in Figura 3. Successivamente sono state applicate le stesse trasformazioni direttamente sulla curva originale ottenendo la B-spline di colore blu mostrata in Figura 3. La proprietà di invarianza per trasformazioni affini viene confermata dal fatto che le due curve combaciano.

Listing 2: Proprietà - Trasformazioni affini

```
num_curve_points = 1000;
knot_vector = [0 0 0 0.3 0.6 1 1 1];
degree = 3;
control_points = [0.1 0.3; 0.05 0.5; 0.2 .5; 0.3 0.8; 0.5 0.6];

% Set the figure window for drawing plots.
fig = figure('Name', 'B-spline Affine Transformations', 'NumberTitle', 'off');
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([0 1]);
ylim([0 1]);

% 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 = bspline_deboor(degree, knot_vector, control_points');
original_curve = curve;
original_curve_plot = plot(curve(1, :), curve(2, :), 'linewidth', 3, ...
                           'color', '#D95319');

% Tranlation, rotation and scaling transformations.
translation = [.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.
curve = bspline_deboor(degree, knot_vector, control_points');
trasf_control_plot = plot(curve(1, :), curve(2, :), 'linewidth', 3, ...
                          'color', 'blue');

% 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', 'yellow');

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');
```

Figura 3: Trasformazioni Affini di una Curva B-spline



2.2.2 Località

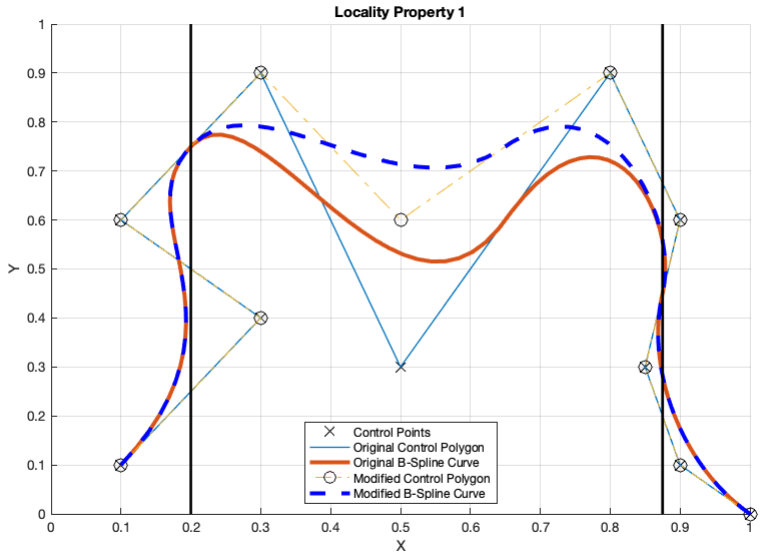
La proprietà di località ci dice che il punto di controllo $\mathbf{d}_i$ influenza la curva solamente per $t \in [t_i, t_{i+k}]$. Negli script 3 e 4 viene mostrata la proprietà di località. Nel primo script (3) viene mostrata come descritta sopra, ovvero data una B-spline di ordine 4 con 10 punti di 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 $\mathbf{d}_{r-k+1}, \ldots, \mathbf{d}_r$. Nello script 4 è mostrato questo comportamento, scegliendo $r = 6$ e modificando i punti di controllo $\mathbf{d}_j \notin [\mathbf{d}_3, \mathbf{d}_6]$ la curva $\mathbf{x}(t^*)$ rimane invariata per $t^* \in [t_6, t_7)$ come si può vedere in Figura 5.

Listing 3: Proprietà - Località 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; 0.85 0.3; 0.9 0.1; 1 0];

% Set the figure window for drawing plots.
fig = figure('Name', 'Locality Property 1', 'NumberTitle', 'off');
```

```
hold on;grid on;
xlabel('X'); ylabel('Y'); title('Locality Property 1');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([0 1]); ylim([0 1]);

% 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 = bspline_deboor(degree, knot_vector, control_points');
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', ...
     'color', '#EDB120', 'MarkerEdgeColor', 'k', 'MarkerSize', 10);

% Calculate and plot the modified B-spline curve.
curve = bspline_deboor(3, knot_vector, control_points');
plot(curve(1, :), curve(2, :), '--', 'linewidth', 3, 'color', 'blue');

% Plot lines for highlighting the modified interval.
left_line = bspline_deboor(degree,knot_vector,control_points',...
                           knot_vector(control_point_mod));
right_line = bspline_deboor(degree,knot_vector,control_points',...
                           knot_vector(control_point_mod+degree+1));
plot([left_line(1) left_line(1)], [0 1], 'k', 'linewidth', 2)
plot([right_line(1) right_line(1)], [0 1], 'k', 'linewidth', 2)

legend({'Control Points', 'Original Control Polygon', ...
        'Original B-spline Curve', 'Modified Control Polygon', ...
        'Modified B-spline Curve'}, 'Location', ...
        'south');
```

Figura 4: B-spline proprietà di Località - Esempio 1



Listing 4: Proprietà - Località 2

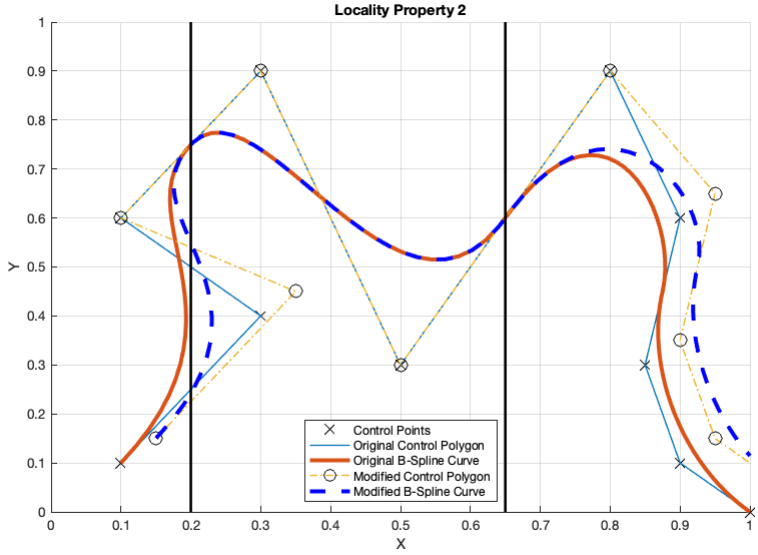
```
num_curve_points = 1000;
knot_vector = [0 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1 1];
degree = 3;
control_points =   [0.1 0.1; 0.3 0.4; 0.1 0.6; 0.3 0.9; 0.5 0.3; 0.8 0.9; ...
                    0.9 0.6; 0.85 0.3; 0.9 0.1; 1 0];

% Set the figure window for drawing plots.
fig = figure('Name', 'Locality Property 2', 'NumberTitle', 'off');

hold on; grid on;
xlabel('X'); ylabel('Y');
title('Locality Property 2');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([0 1]); ylim([0 1]);

% 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 = bspline_deboor(degree, knot_vector, control_points');
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.
curve = bspline_deboor(degree, knot_vector, control_points');
plot(curve(1, :), curve(2, :), '--', 'linewidth', 3, 'color', 'blue');

% Plot lines for highlighting the untouched interval.
left_line = bspline_deboor(degree, knot_vector, control_points', ...
                           knot_vector(r));
right_line= bspline_deboor(degree, knot_vector, control_points', ...
                           knot_vector(r+1));
plot([left_line(1) left_line(1)], [0 1], 'k', 'linewidth', 2)
plot([right_line(1) right_line(1)], [0 1], 'k', 'linewidth', 2)

legend({'Control Points', 'Original Control Polygon', ...
        'Original B-spline Curve', 'Modified Control Polygon', ...
        'Modified B-spline Curve'}, 'Location', ...
        'south');
```

2.2.3 Strong Convex Hull

Ogni punto sulla curva appartiene all'inviluppo convesso di k punti di controllo consecutivi. Nello script 5 viene mostrata la proprietà di Strong Convex Hull per

Figura 5: B-spline proprietà di Località - Esempio 2



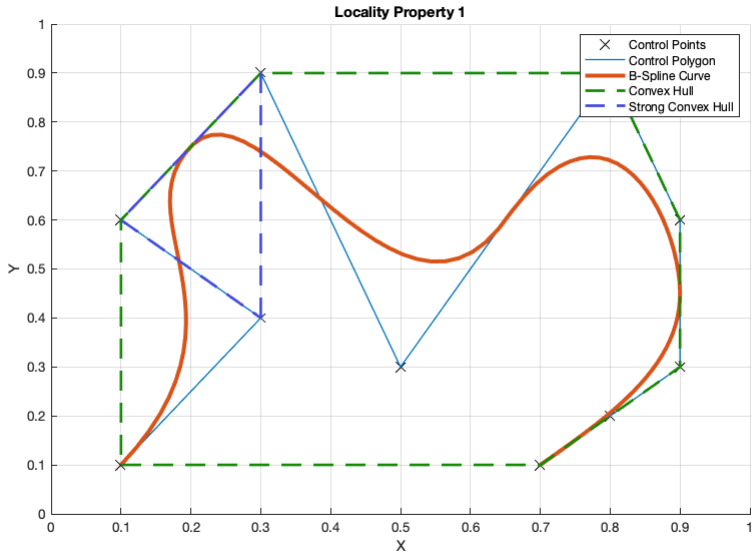
un numero di punti di controllo consecutivi pari a 3, il risultato viene mostrato in Figura 6.

Listing 5: Proprietà - Strong Convex Hull

```
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', 'Strong Convex Hull Property', 'NumberTitle', 'off');
hold on;grid on;
xlabel('X'); ylabel('Y'); title('Locality Property 1');
axes = gca;
axes.XAxisLocation = 'origin';
axes.YAxisLocation = 'origin';
xlim([0 1]); ylim([0 1]);

% 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 = bspline_deboor(degree, knot_vector, control_points');
plot(curve(1, :), curve(2, :), 'linewidth', 3, 'color', '#D95319');

[k,av] = convhull(control_points);
plot(control_points(k,1),control_points(k,2), '--', 'linewidth', 2,...
    'color', '#148A06')

[k,av] = convhull(control_points(end-5+1:end-2,:));
plot(control_points(k+1,1),control_points(k+1,2), '--', 'linewidth', 2,...
    'color', '#424ADC')

legend({'Control Points', 'Control Polygon', 'B-spline Curve', ...
        'Convex Hull', 'Strong Convex Hull'});
```

Figura 6: B-spline proprietà di Strong Convex Hull - Esempio $k = 3$



2.2.4 Variation Diminishing

La proprietà di Variation Diminishing dice che presa una qualunque retta che interseca un numero b di volte il poligono di controllo, questa intersecherà un numero di volte $a \leq b$ la curva B-spline disegnata a partire dal poligono di controllo. Un esempio realizzato con il Codice 6 è mostrato in Figura 7. 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.

Listing 6: Proprietà - Strong Convex Hull

```
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([0 1]); ylim([0 1]);

% 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 = bspline_deboor(degree, knot_vector, control_points');
plot(curve(1, :), curve(2, :), 'linewidth', 3, 'color', '#D95319');

% Generate 3 random line to intersect with the curve.
for i=1:3
    y = 0.1 + (0.8-0.1).*rand(1, 2);
```

```
31      plot([0 1], y, 'k', 'linewidth', 2, 'color', 'b');
32  end
33
34  legend({'Control Points', 'Control Polygon', 'B-spline Curve',...
35          'Intersecting Lines'}, 'Location', 'best');
```

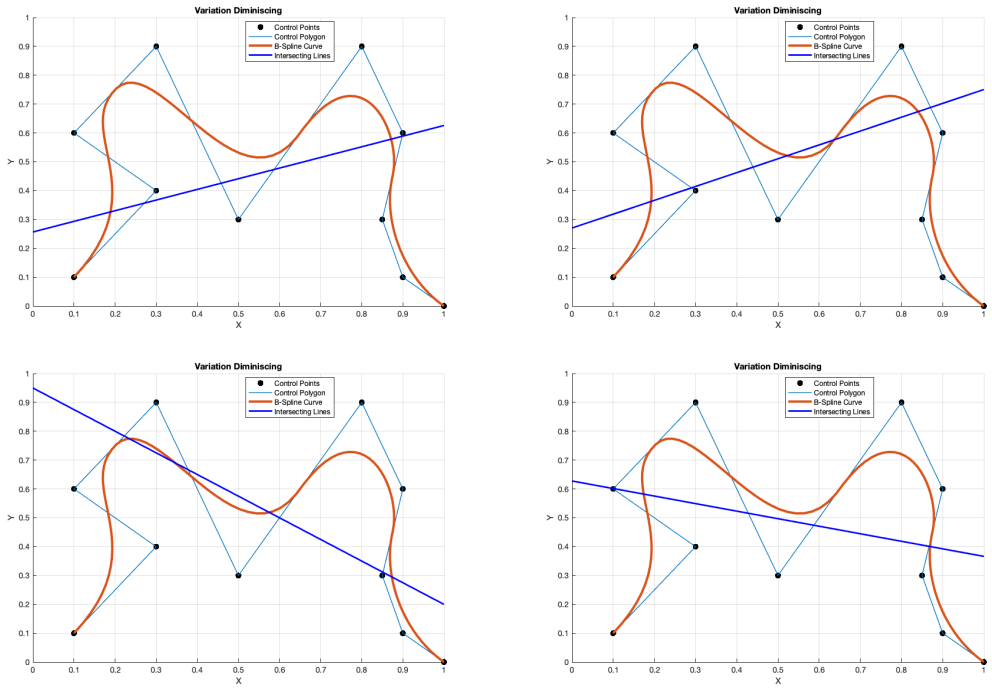


Figura 7: Variation diminishing, quattro esempi di rette

2.3 Rappresentazione di curve B-spline chiuse

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

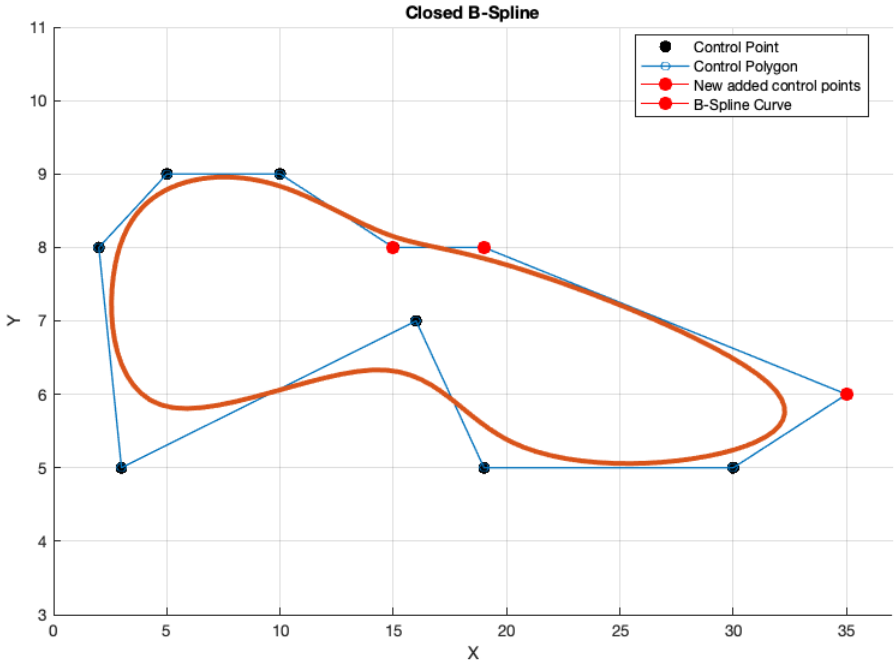
$$\Delta^* = \left[\underbrace{\frac{-k}{m-1}}_{Inizio} : \underbrace{\frac{1}{m-1}}_{Passo} : \underbrace{\frac{k+m-1}{m-1}}_{Fine} \right]$$

con k ordine della spline e m numero di vertici di controllo della curva. Successivamente è anche necessario estendere il poligono di controllo ripetendo i primi $k-1$ vertici. Lo script Matlab 7, 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 8 è riportato un esempio di curva B-spline chiusa di ordine quattro con dieci vertici di controllo generata dallo script 7 .

Listing 7: B-spline - Curva chiuse

```
k = 4;
px = [35 19 15 10 5 2 3 16 19 30 35 19 15];
py = [ 6  8  8  9 9 8 5  7  5  5  6  8  8];
m = (length(px)-k)-1;
knots = -k/m : 1/m : (k+m)/m;

hold on; grid on;
xlabel('X');
ylabel('Y');
title('Closed B-spline');
xlim([min(px)-2 max(px)+2]);
ylim([min(py)-2 max(py)+2]);

plot(px, py, 'k.', 'MarkerSize', 20);
plot(px, py, 'o-', 'linewidth', 1, 'color', '#0072BD');

% Calculate and plot the original B-spline curve using de Boor algorithm.
curve = bspline_deboor(k-1, knots, [px; py]);

for i = 1:k-1
    plot(px(i), py(i), 'ro-', 'markersize', 8, 'MarkerFaceColor','r');
end

plot(curve(1, :), curve(2, :), 'linewidth', 3, 'color', '#D95319');
legend({'Control Point', 'Control Polygon','New added control points',...
        'B-spline Curve'}, 'Location', 'best');
```

Figura 8: Esempio di B-spline Chiusa



3 Superfici

Una superficie può essere rappresentata attraverso una forma *implicita*

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

o in forma *parametrica*:

$$\mathbf{X}(\mathbf{u}) = \mathbf{X}(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix} \in [a, b] \times [c, d] \subset \mathbb{R}^2$$

per rappresentare una superficie servono due parametri, la superficie stessa è l'immagine di $\mathbf{X}(\mathbf{u})$, $\mathbf{u}$ rappresenta le due coordinate parametriche definite nel dominio parametrico, in questo caso si ha un *patch a topologia rettangolare*. Di seguito viene mostrato un esempio di superficie.

La seguente superfice parametrica

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

è stata implementata in Codice 8. Il plot è visualizzato in Figura 9.

Listing 8: Esempio di una superficie parametrica

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

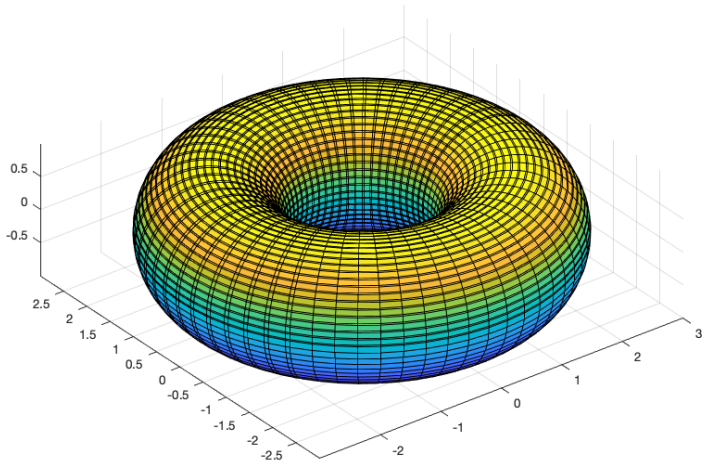


Figura 9: Superficie parametrica

3.1 Superfici tensor-product

Consideriamo due insiemi di funzioni $\{B_0^A(u), \ldots, B_n^A(u)\}$ e $\{B_0^B(v), \ldots, B_m^B(v)\}$ linearmente indipendenti definiti sui due intervalli della retta reale $I_A \in [a_A, b_A]$ e $I_B \in [a_B, b_B]$ con i quali si possono definire i due spazi di funzioni monovariate

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

tali che $\dim S_1 = n+1$ e $\dim S_2 = m+1$. Lo spazio *tensor-product* $S_1 \otimes S_2$ dei due spazi S_1, S_2 è lo spazio delle funzioni bivariate definite sul rettangolo $R = I_A \times I_B$

che sono combinazioni lineari delle funzioni prodotto $B_i^A(u)B_j^j(v)$ al variare di i da 0 a n e j da 0 a m.

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

con $c_{ij} \in \mathbb{R},\ i = 0, \ldots, n,\ j = 0, \ldots, m.$

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 $U = \{u_0, \ldots, u_{n+k}\}, V = \{v_0, \ldots, v_{m+l}\}$ 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

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

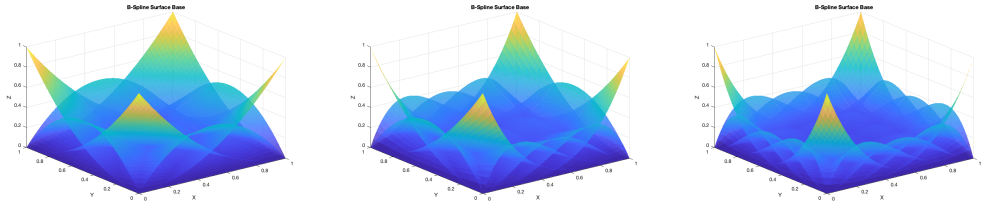
con $(u,v) \in [a,b] \times [c,d]$ è definita a partire da $(n+1)(m+1)$ punti di controllo di de Boor $\mathbf{d}_{i,j}, i = 0, \ldots, n, j = 0, \ldots, m$.

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 10 vengono mostrati alcuni esempi di basi B-spline.

Listing 9: Base delle superfici B-spline

```
% Retrive inputs.
order_1 = 3;
order_2 = 3;
knot_vector_1 = [zeros(1, order_1), ones(1, order_1)];
knot_vector_2 = [zeros(1, order_2), ones(1, order_2)];
num_steps = 50;

% 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) = bspline_basis(j-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) = bspline_basis(j-1, order_2, knot_vector_2,...
                                          steps_1(i));
    end
end


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

% Plot the B-spline surface.
for i = 1 : num_base1_elements
    for j = 1 : num_base2_elements
        Z = first_base(:, i) * transpose(second_base(:, j));
        surf(steps_1 , steps_2 , Z, 'FaceAlpha', 0.8);
        shading flat; s.EdgeColor = 'none';
        hold on;
    end
end
grid on;
xlabel('X');
```



(a) k_u: 3, k_v: 3 (b) k_u: 5, k_v: 5 (c) k_u: 3, k_v: 5

Figura 10: Base delle superfici B-spline

```
ylabel('Y');
zlabel('Z');
title('B-spline Surface Base');
```

Nello script 10 e relativa Figura 11 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:

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

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

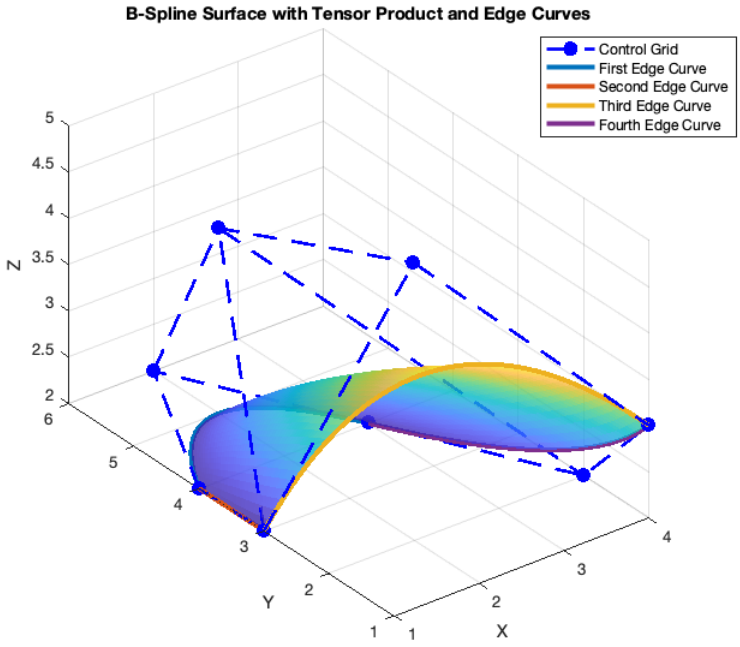
Listing 10: Base delle superfici B-spline

```
control_grid_x = [1 2 3; 1 2 4; 1 2 4];
control_grid_y = [4 6 4; 3 5 2; 3 2 1];
control_grid_z = [2 2 2; 2 4 2; 2 5 3];
order_1 = 3;
order_2 = 3;
knot_vector_1 = [zeros(1, order_1), ones(1, order_1)];
knot_vector_2 = [zeros(1, order_2), ones(1, order_2)];
num_steps = 50;

% 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) = bspline_basis(j-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) = bspline_basis(j-1, order_2, knot_vector_2,...
                                         steps_1(i));
    end
end

% Set the figure window for drawing plots.
fig = figure('Name', 'B-spline Surface with Tensor Product');

% Calculate tensor product and plot the B-spline surface.
second_base_t = transpose(second_base);
surf_x = first_base * control_grid_x * second_base_t;
surf_y = first_base * control_grid_y * second_base_t;
surf_z = first_base * control_grid_z * second_base_t;
surf(surf_x , surf_y , surf_z, 'FaceAlpha', 0.8);
shading flat; s.EdgeColor = 'none';
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, 'Marker
plot3(control_grid_x', control_grid_y', control_grid_z', 'b--', 'linewidth', 2);

% Plot edge curves and legend.
set(axes, 'ColorOrder', circshift(get(gca, 'ColorOrder'), -1))
edge_curve1_plot = plot3(surf_x(1, :), surf_y(1, :), surf_z(1, :));
edge_curve2_plot = plot3(surf_x(:, 1), surf_y(:, 1), surf_z(:, 1));
edge_curve3_plot = plot3(surf_x(end, :), surf_y(end, :),  surf_z(end, :));
edge_curve4_plot = plot3(surf_x(:, end), surf_y(:, end),  surf_z(:, end));
axis tight; axis equal;
legend([pol_plot(1) edge_curve1_plot edge_curve2_plot edge_curve3_plot ...
        edge_curve4_plot], {'Control Grid', 'First Edge Curve', ...
        'Second Edge Curve', 'Third Edge Curve', 'Fourth Edge Curve'}, ...
        'Location', 'best');
```

Figura 11: Superficie B-spline e relative curve di bordo



3.2 Proprietà di invarianza per trasformazioni affini

Come detto in precedenza questa proprietà dice che applicare una trasformazione affine sui punti di controllo o sulla superfice è indifferente. Questa proprietà risulta molto utile nella situazione in cui si deve applicare una determinata trasformazione ad una superficie. Nello script Matlab 15 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 12 . La proprietà di invarianza per trasformazioni affini viene confermata dal fatto che le due superfici trasformate combaciano.

Listing 11: Trasformazione affine superficie B-splinee

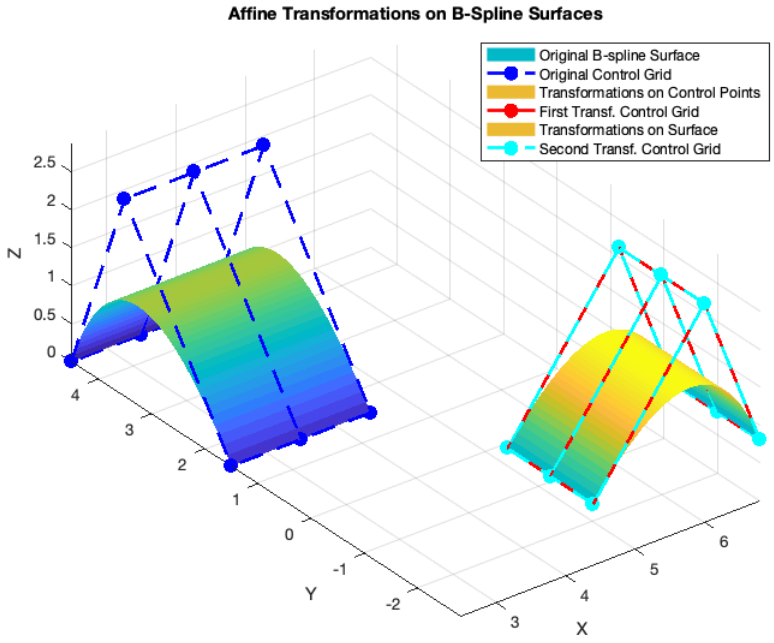
```
control_grid_x = [4.5 3.5 2.5; 4.5 3.5 2.5; 4.5 3.5 2.5];
control_grid_y = [4.5 4.5 4.5; 3.5 3.5 3.5; 1.5 1.5 1.5];
control_grid_z = [0 0 0; 2.6 2.6 2.6; 0 0 0];
order_1 = 3;
order_2 = 3;
knot_vector_1 = [0 0 0 1 1 1];
knot_vector_2 = [0 0 0 1 1 1];
num_steps = 50;

% 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) = bspline_basis(j-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) = bspline_basis(j-1, order_2, knot_vector_2,...
                                          steps_1(i));
    end
end

% Set the figure window for drawing plots.
fig = figure('Name', 'Affine Transformations on B-spline Surface');

% Plot the original B-spline surface.
second_base_t = transpose(second_base);
surf_x = first_base * control_grid_x * second_base_t;
surf_y = first_base * control_grid_y * second_base_t;
surf_z = first_base * control_grid_z * second_base_t;

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.--');
plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'b--');

% 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.
```

```
68  control_points = (control_points*rotation + translation)*scaling;
69
70  % Restore control points in three matrices (control grid).
71  control_grid_x = reshape(control_points(:, 1), order_1, order_2).';
72  control_grid_y = reshape(control_points(:, 2), order_1, order_2).';
73  control_grid_z = reshape(control_points(:, 3), order_1, order_2).';
74
75  % Plot the transformed B-spline surface.
76  surf_x_trans = first_base*control_grid_x*second_base.';
77  surf_y_trans = first_base*control_grid_y*second_base.';
78  surf_z_trans = first_base*control_grid_z*second_base.';
79  trasf_control_plot = surf(surf_x_trans, surf_y_trans, surf_z_trans);
80
81  % Plot the control grid of the B-spline surface.
82  trasf_pol_plot = plot3(control_grid_x, control_grid_y, control_grid_z, 'r.-');
83  plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'r-');
84
85  % Transform surface points matrices into a single matrix for applying
86  % transformations (num_control_points x dimensions).
87  surface_points =   [reshape(surf_x.', [], 1), reshape(surf_y.', [], 1), ...
88                      reshape(surf_z.', [], 1)];
89
90  % Transformation on surface points.
91  surface_points = (surface_points * rotation + translation) * scaling;
92
93  % Restore surface points in three matrices (surf).
94  surf_x = reshape(surface_points(:, 1), num_steps, num_steps).';
95  surf_y = reshape(surface_points(:, 2), num_steps, num_steps).';
96  surf_z = reshape(surface_points(:, 3), num_steps, num_steps).';
97
98  % Plot the transformed B-spline surface.
99  trasf_surf_plot = surf(surf_x , surf_y , surf_z, 'FaceAlpha', 0.8);
100 shading flat; s.EdgeColor = 'none';
101
102 % Plot the control grid of the B-spline surface and legend.
103 trasf_surf_pol_plot = plot3(control_grid_x, control_grid_y, control_grid_z, 'c.--');
104 plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'c--');
105 axis tight; axis equal;
106 legend([origin_surf_plot origin_pol_plot(1) trasf_control_plot ...
107         trasf_pol_plot(1) trasf_surf_plot ...
```

```
108         trasf_surf_pol_plot(1)], {'Original B-spline Surface', ...
109         'Original Control Grid', 'Transformations on Control Points',...
110         'First Transf. Control Grid', 'Transformations on Surface', ...
111         'Second Transf. Control Grid'}, 'Location', 'best');
```

Figura 12: Risultato di una trasformazione affine applicata ad una superficie B-spline



3.3 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:

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

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

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

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

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

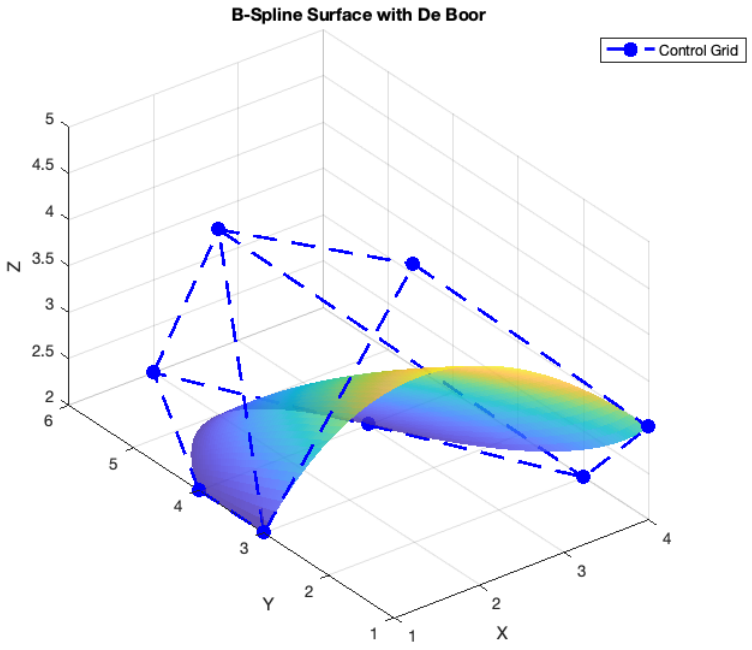
Possiamo quindi utilizzare nuovamente l'algoritmo di de Boor.
Riassumendo, quello che si fa è applicare $n + 1$ volte l'algoritmo di de Boor per calcolare i vari $q_i(v)$ e poi una volta per calcolare $p(u, v)$. Nello script 12 è stata rappresentata mediante l'algoritmo di de Boor la superficie B-spline vista in precedenza. Il risultato di quest'ultimo è mostrato in figura 17.

Listing 12: Trasformazione affine superficie B-splinee

```
% Retrive inputs.
p_x = [1 2 3; 1 2 4; 1 2 4];
p_y = [4 6 4; 3 5 2; 3 2 1];
p_z = [2 2 2; 2 4 2; 2 5 3];
order_1 = 3;
order_2 = 3;
knot_vector_1 = [zeros(1, order_1), ones(1, order_1)];
knot_vector_2 = [zeros(1, order_2), ones(1, order_2)];
num_steps = 50;

control_points = [reshape(p_x', [], 1), reshape(p_y', [], 1), ...
                  reshape(p_z', [], 1)];

% 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);

% Calculate with de boor every point of the B-spline surface.
surface_points = zeros(num_steps*num_steps, 3);
count = 1;
for i = 1 : num_steps
    for j = 1 : num_steps
        % n times de Boor algorithm to calculate n points.
```

```
        n = length(knot_vector_1) - order_1;
        m = length(knot_vector_2) - order_2;
        Q = zeros(n, 3);
        for k = 1 : n
            Q(k, :) =  bspline_deboor(order_2-1, knot_vector_2, ...
                            control_points(m*(k-1)+1: m*k, :)', steps_2(j));
        end

        % de Boor algorithm on Q to calculate the final surface point.
        surface_points(count, :) = bspline_deboor(order_1-1, knot_vector_1,...
                                        Q', steps_1(i));
        count = count + 1;
    end
end

% Set the figure window for drawing plots.
figure('Name', 'B-spline Surface with De Boor', 'NumberTitle', 'off');

% Plot the B-spline surface.
surface_matrix_x = reshape(surface_points(:, 1), num_steps, num_steps).';
surface_matrix_y = reshape(surface_points(:, 2), num_steps, num_steps).';
surface_matrix_z = reshape(surface_points(:, 3), num_steps, num_steps).';
surf(surface_matrix_x, surface_matrix_y, surface_matrix_z, 'FaceAlpha', 0.8);
shading flat; s.EdgeColor = 'none';
hold on; grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('B-spline Surface with De Boor');

% Puts 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 control grid of the B-spline surface.
pol_plot = plot3(control_grid_x, control_grid_y, control_grid_z, 'b.--');
plot3(control_grid_x.', control_grid_y.', control_grid_z.', 'b--');
axis tight; axis equal;
legend(pol_plot(1), {'Control Grid'}, 'Location', 'best');
```

Figura 13: Rappresentazione superficie B-spline mediante algoritmo di de Boor



4 Approssimazione ai minimi quadrati

Supponiamo di avere N punti ricavati sperimentalmente e quindi soggetti a errori (x_i, f_i), con $i = 1, \ldots, N$, $x_i \neq x_j$ per $i \neq j$. Supponendo che tra le x_i e le f_i esista una certa relazione, vogliamo trovare la curva di grado n (con $n << N$[1]) che approssima meglio questi dati. La funzione che descrive la curva appartiene a uno spazio di dimensione $n + 1$, span$\{\phi_0(x) \ldots, \phi_n(x)\}$ definito in un intervallo $[a, b]$ dove i ϕ_i sono gli elementi di una generica base dello spazio scelto, ad esempio se vogliamo un'approssimazione polinomiale possiamo scegliere come base la base canonica e allora avremo $\phi_i(x) = x^i$.

In generale la funzione cercata sarà espressa come

$$f(x) = \sum_{i=0}^{n} a_i \phi_i(x)$$

[1] se $n = N$ si avrebbe interpolazione.

L'approssimazione ai minimi quadrati prevede di minimizzare la distanza quadratica[2] della curva dai punti, nella direzione dell'asse y, cioè di trovare i valori $a_0, \ldots, a_n$ che minimizzano la funzione

$$d(a_0, \ldots, a_n) = \sum_{i=1}^{N} \left(f_i - \sum_{j=0}^{n} a_j \phi_j(x_i) \right)^2$$

il minimo di questa funzione si ottiene quando il gradiente è 0 cioè quando tutte le derivate parziali rispetto a $a_0, \ldots, a_n$ sono nulle

$$\frac{\partial}{\partial a_k} d(a_0, \ldots, a_k) = 0, \quad k = 0, \ldots, n$$

4.1 Implementazione

Per risolvere in maniera efficiente il problema di minimo

$$\min_{a_0,\ldots,a_n} \sum_{i=1}^{N} \left(f_i - \sum_{j=0}^{n} a_j \phi_j(x_i) \right)^2$$

notiamo che definendo

$$A = \begin{pmatrix} \phi_0(x_1) & \ldots & \phi_n(x_1) \\ \vdots & \ddots & \vdots \\ \phi_0(x_N) & \ldots & \phi_n(x_N) \end{pmatrix} \qquad \mathbf{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \qquad \mathbf{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

La matrice A si dice matrice di *collocazione*. Si definisce quindi $\mathbf{f} - A\mathbf{a}$ e si riscrive la relazione di minimo come

$$\min_{\mathbf{a}} ||\mathbf{f} - A\mathbf{a}||_2^2$$

Utilizzando il metodo di Householder[3] posso trovare una matrice Q ortogonale tale che $A = QR$ e, poiché moltiplicare per una matrice ortogonale non cambia la norma 2 si ha che

$$\begin{aligned} \min_{\mathbf{a}} \|\mathbf{f} - A\mathbf{a}\|_2^2 &= \min_{\mathbf{a}} \left\| Q^T f - Q^T A\mathbf{a} \right\|_2^2 \\ &= \min_{\mathbf{a}} \left\| \tilde{f} - R\mathbf{a} \right\|_2^2 && (Q^T f = \tilde{f}) \\ &= \min_{\mathbf{a}} \left\| \begin{bmatrix} \tilde{f}_1 - R\mathbf{a} \\ \tilde{f}_2 \end{bmatrix} \right\|_2^2 && (\tilde{f} = (\tilde{f}_1, \tilde{f}_2)^T) \\ &= \min_{\mathbf{a}} \left\| \tilde{f}_1 - R_1\mathbf{a} \right\|_2^2 + \left\| \tilde{f}_2 \right\|_2^2 \end{aligned}$$

[2] non la distanza euclidea, in quanto si dovrebbe considerare la distanza fra il punto e la curva, misurata come la lunghezza della retta perpendicolare al vettore tangente.

[3] Per le trasformazioni affini, non si modifica la lunghezza del vettore se la matrice Q è ortogonale, infatti $\|Qv\|_2^2 = v^T Q^T Q v = \|v\|_2^2$.

Ma poiché si deve minimizzare rispetto ad $\mathbf{a}$ il secondo termine può essere ignorato e si deve cercare il $\min_{\mathbf{a}} \left\| \tilde{f}_1 - R_1\mathbf{a} \right\|_2^2$ che si ottiene per $\tilde{f}_1 - R_1\mathbf{a} = 0$ quindi la soluzione si ottiene risolvendo il sistema lineare triangolare (grazie alla proprietà della fattorizzazione QR) $R\mathbf{a} = \tilde{f}_1$. Nel nostro caso inoltre la funzione approssimante è descritta prametricamente quindi dovremo associare in qualche modo ad ogni punto $\mathbf{P}$ un parametro t_i e poi risolvere tanti problemi ai minimi quadrati quante sono le dimensioni dello spazio a cui appartiene la curva, minimizzano quindi la distanza tra i punti $\mathbf{P}_i$ e $\mathbf{X}(t_i)$.

Se le ϕ sono delle B-splines, si ha lo spazio $V = \text{span}\{N_{0,k}(t), \ldots, N_{n,k}(t)\}$ e il vettore esteso dei nodi $\mathbf{t} = \{t_0, \ldots, t_{n+k}\}$, dunque la seguente matrice rettangolare

$$A = \begin{pmatrix} N_{0,k}(t_1) & \ldots & N_{n,k}(t_1) \\ \vdots & \ddots & \vdots \\ N_{0,k}(t_N) & \ldots & N_{n,k}(t_N) \end{pmatrix}$$

La funzione che si ottiene sostituendo ad una base generica quella delle B-spline è la seguente:

$$f(x) = \sum_{i=1}^{n+k} a_i N_{i,k}(t)$$

Il procedimento ora risulta simile a quello visto in precedenza. L'obiettivo è sempre quello di determinare i coeffcienti a_i in maniera tale da avere una curva risultante che approssima ai minimi quadrati i punti dati, ciò equivale a risolvere il sistema lineare $A^T A\mathbf{a} = A^T\mathbf{y}$ tramite fattorizzazione QR della matrice A. Dobbiamo tenere sempre a mente che le B-spline $N_{i,k}(t)$ dovranno sempre essere valutate nell'intervallo opportuno. Al momento della creazione della matrice A dobbiamo, per ogni punto t_i, stabilire qual è il suo supporto su $\mathbf{t}$ e qui valutare la funzione. Nello script Matlab 14 viene mostrato il codice che si occupa di approssimare nel senso dei minimi quadrati un insieme di punti x. La funzione `bs_least_square` prende in ingresso: il vettore dei punti $\{x, y\}$, il grado della curva spline da adattare e il vettore dei nodi.

Listing 13: Approssimazione ai minimi quadrati

```
1  function [fitted, control_points] = bs_least_square(x, y, d, knots)
2  % bs_least_square:
3  %   Implementation of Least Square approximation by using B-Spline basis,
4  %   return the fitted curve on data poins [x, y].
5  %
6  % Syntax: [y_fit, err] = bs_least_square(x, y, d, knots, 0.005);
7  %
```

```
% Input:
%   - x: vector of x points.
%   - y: vector of y points.
%   - d: order of the B-Spline base.
%   - knots: knot vector of the B-Spline curve.
%

XData = [x y];

% discrete parametrization
e = 1;
n = size(XData,1);
u = zeros(n,1);
u(1)=0;

nominator = 0;
denominator = 0;

for j = 1:n-1
    denominator = denominator + (norm(XData(j+1,:)-XData(j,:)))^e;
end

for i = 2:n
    j=i-1;
    nominator = nominator + (norm(XData(j+1,:)-XData(j,:)))^e;

    nextU=nominator/denominator;
    u(i)=nextU;
end

% get B-spline basis
m=numel(u);
B=zeros(m,numel(knots)- d);
for i = 1 : numel(knots)- d
    for j = 1:m
        B(j,i) = bspline_basis(i-1,d,knots, u(j));
    end
end

A = B' * B;
```

```
48
49  % Estimate control points for x
50  b = B' * x;
51  Cx = QR_solve(A,b);
52
53  % Estimate control points for y
54  b = B' * y;
55  Cy = QR_solve(A,b);
56
57  fitted = B * Cy;
58
59  control_points=[Cx Cy];
60
61  end
```

Listing 14: Approssimazione ai minimi quadrati - fattorizzazione QR e risoluzione del sistema triangolare superiore associato.

```
1   function [fit] = QR_solve(A, b)
2
3   [houseQ, houseR] = qr(A);
4
5   householdery = houseQ' * b;
6
7   % Perform backward substitution
8   % Store the dimensions of the upper triangular U
9   [m,n]=size(houseR);
10
11  % Initiate the zero colum vector
12  fit=zeros(m,1);
13
14  % Iterate over the rows
15  for j=m:-1:1
16      % Compute the j-th entry
17      fit(j) = (householdery(j) - houseR(j, :)*fit) / houseR(j, j);
18  end
19
20  end
```

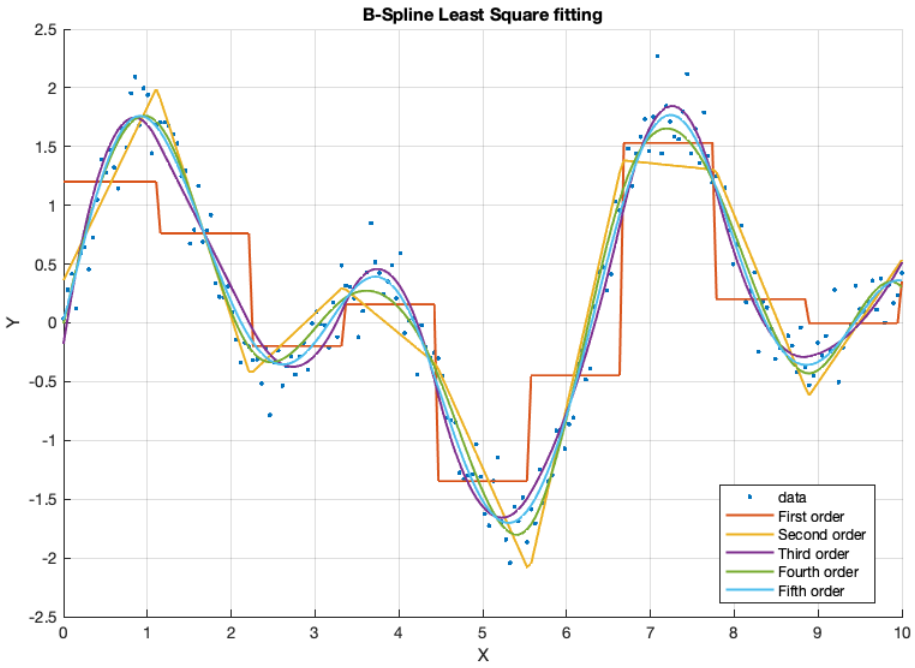
Listing 15: Approssimazione ai minimi quadrati - Esempio

```
% setup
d = 3;

f = @(x) sin(x) + sin(2*x);
f_err = @(x) f(x) + 0.2 * randn(size(x));

xMin = 0;
xMax = 10;
nx = 200;
nknots = 10;

x = linspace(xMin, xMax, nx)';
y = f(x);
y_err = f_err(x);
knots = linspace(xMin, xMax, nknots);

% plot
hold on;grid on;
xlabel('X');ylabel('Y');
title('B-spline Least Square fitting');
plot(x, y_err, '.', 'DisplayName', 'data');
%plot(x, y);

for d=1:5
    [y_fit, c] = bs_least_square(x, y, d, knots);

    ordinal = iptnum2ordinal(d);
    plot(x, y_fit, 'linewidth', 1.5, 'DisplayName', [upper(ordinal(1)),...
        ordinal(2:min(end)) ' order']);
end

% stat
aver = sum(y)/nx;
rrmse = sqrt(sum((y - y_fit).^2)/nx) / aver * 100;

legend('Location', 'best');
```

In Figura 14 si vedono le curve ottenute approssimando ai minimi quadrati i dati sperimentali simulati attraverso funzioni casuali ma con un andamento definito.

In Figura 15 viene mostrata la curva B-spline di grado 5 con i relativi punti e poligono di controllo relativa agli stessi dati simulati.

Figura 14: Approssimazione ai minimi quadrati, B-spline di grado $k = 1, ..., 5$



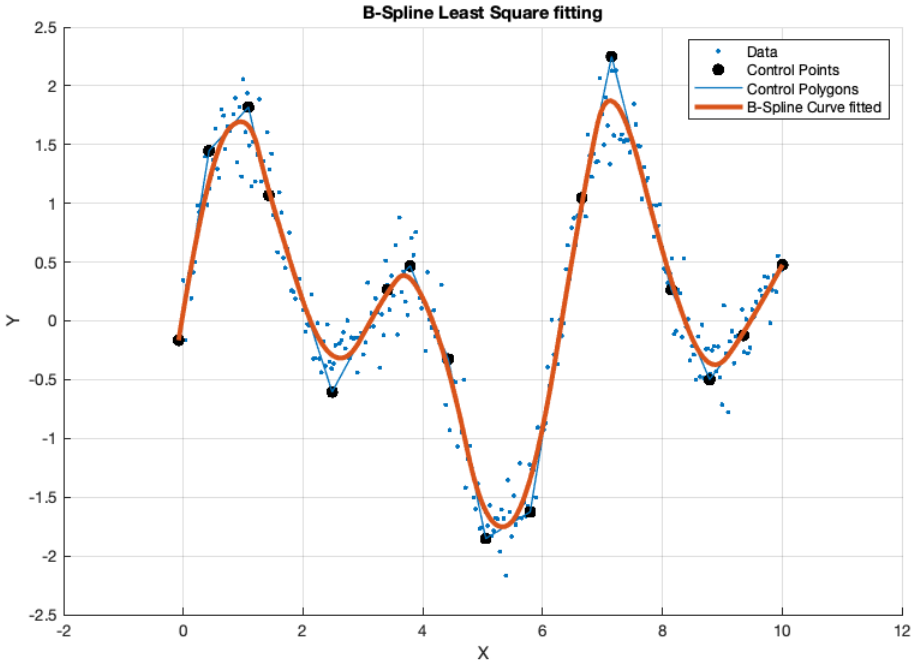
4.2 Applicazione al caso reale: mortalità regionale Covid-19

In questa sezione vediamo come possiamo approssimare i dati discreti, forniti dal Dipartimento della Protezione Civile[4], sulla mortalità regionale Covid-19 in modo da adattare curve B-spline ai dati in questione.

Dal 24 febbraio 2020, il Dipartimento della Protezione Civile su base giornaliera, pubblica quotidianamente il numero cumulativo di decessi Covid-19 divisi per ciascuna delle 21 regioni italiane. Il database in questione è denominato `dpc-covid19-ita-regioni.json`, tale dataset elenca, per ogni regione, le principali informazioni sulle province (codice, denominazione, sigla, latitudine e longitudine), sulla regione (codice e denominazione) e, infine, fornisce informazioni sul totale dei casi rilevati e la data di rilevazione. Prima d'iniziare il lavoro di adattamento, sulle variabili d'interesse verranno applicati dei semplici metodi di ripulitura e normalizzazione, in modo da preparare le stesse alla successiva fase. In

[4]Il dataset è consultabile all'indirizzo `https://github.com/pcm-dpc/COVID-19`

Figura 15: Approssimazione ai minimi quadrati, B-spline di grado $k = 5$



particolare, essendo il dato riguardante la mortalità ragionale di tipo cumulativo, il suo valore aumenta ogni qual volta un nuovo decesso viene registrato, l'obiettivo è quello di trasformare il dato cumulato in un dato giornaliero effettivo. Per fare questo viene utilizzato un semplice script Python, applicato nel range di date che vanno dal 25/02/2020 al 01/02/2021.

Sia m il numero di giorni disponibili nel dataset e n il numero di regioni, definiamo $\boldsymbol{D} = (d_{s,r})$ la matrice $m \times n$ contenente i decessi osservati al tempo $s = 1, ..., m$ per ogni regione $r = 1, ..., n$. Per ogni regione r si adatta una curva approssimata nel senso dei minimi quadrati ai dati $\boldsymbol{D}$ che minimizza la funzione

$$f(x) = \sum_{s=0}^{m} \left(d_{s,r} - \sum_{j=0}^{N} a_j N_{j,k}(t) \right)^2$$

Nel listato 16 è mostrato il codice per ottenere tali curve per ognuna delle 21 regioni Italiane.

Listing 16: Approssimazione ai minimi quadrati - Esempio

```
T = readtable('datasets/dataCOVID_all.csv');
[G, id] = findgroups(T.region);

tiledlayout(7,3)

% setup
d=3;
nknots = 30;
lambda=.5; % smoothing parameter

% just for plotting purpose, start date is 25 feb = (1 Jan) + 55 days
offset = 55;

for i = 1:length(id)
    g = T(string(T.region)==id{i}, :);

    x = (offset:1:size(g,1)+offset-1)';
    y = g.deaths;
    y(y<0) = 0;

    knots = linspace(0, size(g,1), nknots);
    [y_fit, ~] = bs_least_square(x, y, d, knots, lambda);

    ax=nexttile;
    title(ax,id{i})
    hold on;grid on;
    plot(x, y, '.', 'DisplayName', 'data');
    plot(x, y_fit, 'linewidth', 1.5);
    datetick('x','mmm')
end
```

4.3 Superfici

Possiamo sfruttare i concetti esposti nel paragrafo precedente per disegnare superfici che approssimano nel senso dei minimi quadrati un insieme di punti tridimensionale. Sia $(r_{k_0}, s_{k_1}, \mathbf{P}_{k_0k_1})$ un insieme di punti con $0 \leq k_0 \leq m_0$ e $0 \leq k_1 \leq m_1$. Si assume che $r_0 < r_1 < \cdots < r_{m_0}$ e $s_0 < s_1 < \cdots < r_{m_1}$. Una superficie B-spline che approssima tali punti è parametrizzata da $(u, v) \subset \mathbb{R}^2$, dunque i punti

r e s devono essere mappati sul dominio parametrico $u_{k_0} = (r_{k_0} - r_0)/(r_{m_0} - r_0)$ e $v_{k_1} = (s_{k_1} - s_0)/(s_{m_1} - s_0)$.

La superficie B-spline approssimata è sostanzialmente la seguente

$$\mathbf{X}(u, v) = \sum_{i=0}^{n_0} \sum_{j=0}^{n_1} \mathbf{d}_{i,j} N_{i,d_0}(u) N_{j,d_1}(v)$$

con d_0, d_1 ordine della rispettiva base, i punti di controllo $\mathbf{d}_{i,j}$ sono quantità sconosciute da determinare. I punti di controllo possono essere organizzati formalmente come una matrice $(n_0 + 1) \times (n_1 + 1)$,

$$\hat{D} = \begin{pmatrix} \mathbf{d}_{00} & \dots & \mathbf{d}_{0m} \\ \vdots & \ddots & \vdots \\ \mathbf{d}_{n0} & \dots & \mathbf{d}_{nm} \end{pmatrix}$$

Allo stesso modo, i $\mathbf{P}_{k_0 k_1}$ possono essere disposti in una matrice di dimensione $(m_0 + 1) \times (m_1 + 1)$ nel modo seguente,

$$\hat{P} = \begin{pmatrix} \mathbf{P}_{00} & \dots & \mathbf{P}_{0m_1} \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{m_0 0} & \dots & \mathbf{P}_{m_0 m_1} \end{pmatrix}$$

Per un insieme specifico di punti di controllo, l'approssimazione consiste nel minimizzare la distanza quadratica tra la superficie B-spline e i punti $\mathbf{P}_{k_0 k_1}$, ovvero minimizzare la funzione obiettivo

$$F(\hat{D}) = \sum_{k_0=0}^{m_0} \sum_{k_1=0}^{m_1} \left(\mathbf{P}_{k_0 k_1} - \sum_{i=0}^{n_0} \sum_{j=0}^{n_1} \mathbf{d}_{i,j} N_{i,d_0}(u_{k_0}) N_{j,d_1}(v_{k_1}) \right)^2$$

La funzione F è quadratica nelle componenti di $\hat{D}$, il suo grafico è un paraboloide, quindi ha minimo globale quando tutte le sue derivate parziali del primo ordine sono nulle. Le derivate parziali del primo ordine sono scritte rispetto ai punti di controllo $\mathbf{d}_{i,j}$ piuttosto che in termini di componenti dei punti di controllo, siano A e B le matrici rettangolari relative alle basi B-spline $N_{i,d_0}(u)$ e $N_{j,d_1}(v)$ rispettivamente, possiamo scrivere la derivata parziale di F rispetto a $\hat{D}$ come

$$\frac{\partial F}{\partial \hat{D}} = A^T A \hat{D} B^T B - A^T \hat{P} B$$

dove $\partial F / \partial \hat{D}$ è una matrice $(n_0 + 1) \times (n_1 + 1)$, $A = [a_{rc}]$ è una matrice $(m_0 + 1) \times (n_0 + 1)$ e $B = [b_{rc}]$ è una matrice $(m_1 + 1) \times (n_1 + 1)$.

Impostando le derivate parziali uguali alla matrice nulla $\hat{\mathbf{0}}$ si ottiene il sistema matriciale di equazioni,

$$A^T A\hat{D}B^T B - A^T\hat{P}B = \hat{\mathbf{0}}$$

La matrice $A^T A$ è simmetrica e a *banda*. Una matrice a banda è una matrice sparsa i cui elementi diversi da zero sono tutti posti in una banda diagonale che comprende la diagonale principale e, opzionalmente, una o più diagonali alla sua destra od alla sua sinistra. Nel nostro caso, il numero di bande superiori e il numero di bande inferiori sono gli stessi, cioè $d_0 + 1$. La banda è una conseguenza del controllo locale per le superfici B-spline. Un'analisi simile mostra che anche $B^T B$ è simmetrica e a banda, con $d_1 + 1$ bande superiori e $d_1 + 1$ bande inferiori.

L'approccio diretto per risolvere l'equazione è invertire le matrici $A^T A$ e $B^T B$,

$$\hat{D} = (A^T A)^{-1} A^T \hat{P} B (B^T B)^{-1} = XPY^T$$

si procede all'inversione delle matrici mediante la fattorizzazione QR come nel caso univariato. Nello script Matlab 17 è stata implementata la funzione che si occupa di approssimare punti in $\mathbb{R}^3$ con una superficie. In Figura 18 viene mostrato il risultato della funzione `bs_least_square_2` su un insieme di punti simulato, in particolare si cerca di rappresentare nel modo migliore possibile la funzione $z = \log(4x^2 + y^2)$ con x, y vettori uniformi nei rispettivi intervalli $x \in [-3, 3]$; $y \in [-10, 10]$. In Figura 19 viene mostrata la superficie con i rispettivi punti di controllo calcolati dall'approssimazione.

Listing 17: Approssimazione ai minimi quadrati per superfici

```
function [z_fit, control_points] = bs_least_square_2(x, y, z, d,...
            knots_x, knots_y, lambda)
% bs_least_square:
%   Implementation of Least Square approximation by using B-Spline basis,
%   return the fitted curve on data poins [x, y].
%
% Syntax: [y_fit, err] = bs_least_square(x, y, d, knots, 0.005);
%
% Input:
%   - x: vector of x points.
%   - y: vector of y points.
%   - y: vector of y points.
%   - d: order of the B-Spline base.
%   - knots_x, knots_y: knot vector of the B-Spline curve.
%   - lambda: parameter value.
%
```

```
17
18  if nargin < 7
19          lambda = 0;
20  end
21  x = x(:);
22  y = y(:);
23  z = z(:);
24
25  xmin = knots_x(1);
26  xmax = knots_x(end);
27  tx = [repmat(xmin, [1, d]), knots_x, repmat(xmax, [1, d])];
28
29  ymin = knots_y(1);
30  ymax = knots_y(end);
31  ty = [repmat(ymin, [1, d]), knots_y, repmat(ymax, [1, d])];
32
33  ncoeff_x = numel(knots_x) + d - 1;
34  ncoeff_y = numel(knots_y) + d - 1;
35  B = zeros(numel(x), ncoeff_x*ncoeff_y);
36
37  bspline_x = zeros(numel(x), ncoeff_x);
38  bspline_y = zeros(numel(x), ncoeff_y);
39  parfor j = 1 : ncoeff_x
40      bspline_x(:, j) = bspline_basis(j-1, d+1, tx, x);
41  end
42
43  parfor k = 1 : ncoeff_y
44      bspline_y(:, k) = bspline_basis(k-1, d+1, ty, y);
45  end
46
47  for j = 1 : ncoeff_x
48      for k = 1 : ncoeff_y
49          B(:, (j - 1)*ncoeff_y + k) = bspline_x(:, j).*bspline_y(:, k);
50      end
51  end
52
53  A = B' * B + lambda*eye(ncoeff_x*ncoeff_y);
54
55  % Estimate control points for x
56  b = B' * x;
```

```
57  Cx = QR_solve(A,b);
58
59  % Estimate control points for y
60  b = B' * y;
61  Cy = QR_solve(A,b);
62
63  % Estimate control points for z
64  b = B' * z;
65  Cz = QR_solve(A,b);
66
67
68  % evaluation
69  z_fit = reshape(B * Cz, size(x));
70
71  control_points = [Cx Cy Cz];
72
73  end
```

Figura 16: Approssimazione ai minimi quadrati - Esempio mortalità Covid-19

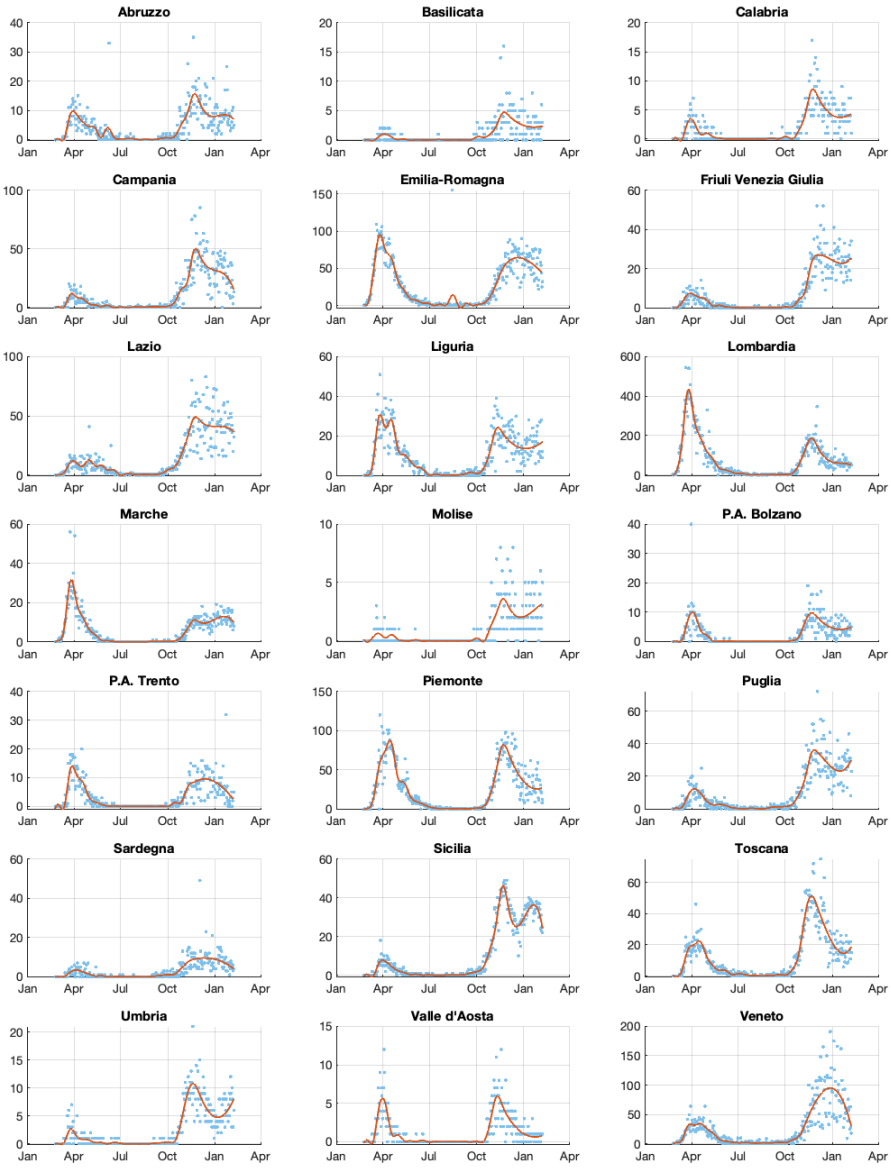


Figura 17: Approssimazione ai minimi quadrati - Esempio mortalità Covid-19 per la regione Toscana

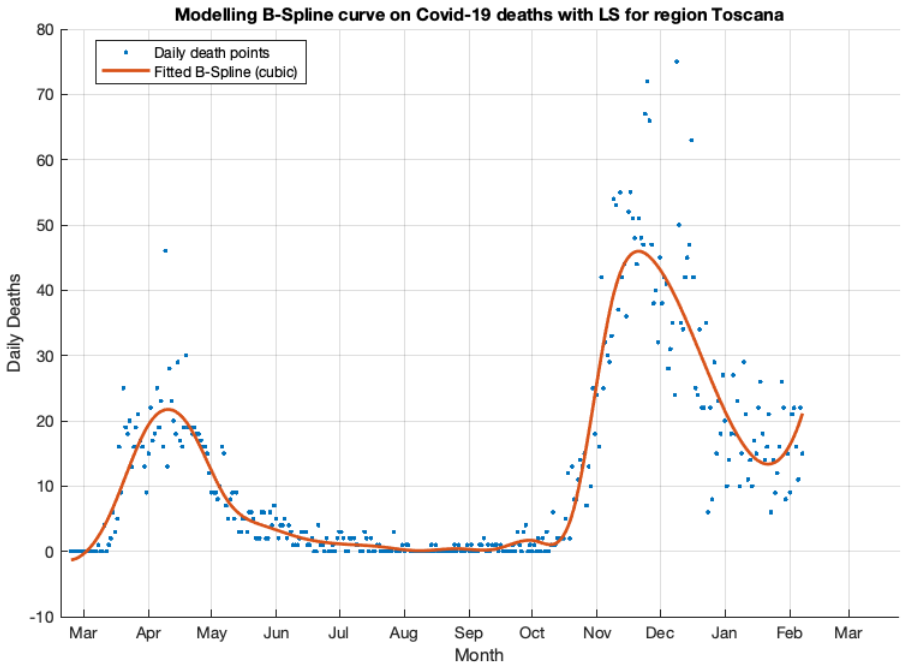


Figura 18: Approssimazione ai minimi quadrati per una superficie, in particolare la superficie $z = \log(4x^2 + y^2)$

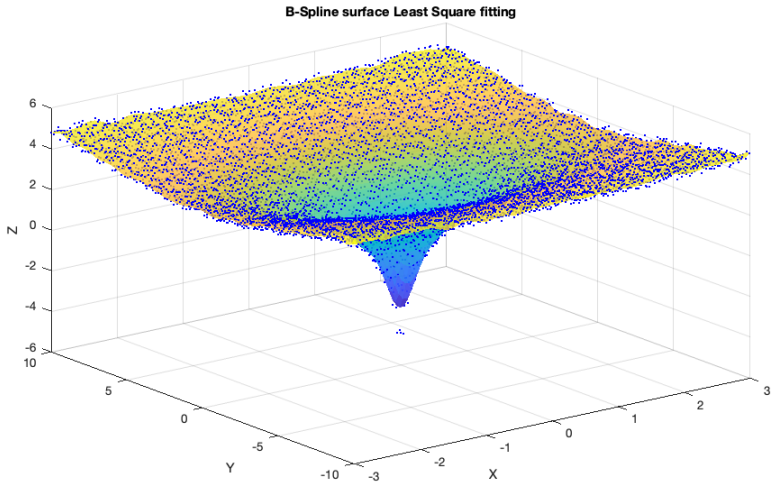


Figura 19: Approssimazione ai minimi quadrati per una superficie, nell'immagine vengono mostrati i punti di controllo.