Metodi Numerici per la Grafica

- Relazione -

Superfici di Bezier e B-Spline

Marco Calamai - Elia Mercatanti

21 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

```
function [y] = cox_de_boor(i, r, t, t_star, k)
    if r == 1
        % Check even the last interval special case.
        if (t_star >= t(i) && t_star < t(i+1)) || ((t_star >= t(i)
            && ...
```

```
                t_star <= t(i+1) && t_star == t(end) && i == length(t)-k))
                y = 1;
            else
                y = 0;
            end
        else
            y = omega(i, r, t_star, t)*cox_de_boor(i, r-1, t, t_star, k) + ...
                (1 - omega(i+1, r, t_star, t)) * ...
                cox_de_boor(i+1, r-1, t, t_star, k);
    end
end

function [omega_ir] = omega(i, r, t_star, t)
    if t(i) == t(i+r-1)
        omega_ir = 0;
    elseif t_star <= t(i+r-1)
        omega_ir = (t_star-t(i)) / (t(i+r-1)-t(i));
    else
        omega_ir = 0;
    end
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:
%
% 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 input.
```

```
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});
t = str2num(inputs{2});
num_points = 1000;

% 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';

% Initialization
steps = linspace(t(1), t(end), num_points);
base_y = zeros(1, num_points);

for i = 1 : length(t) - order
    for j = 1 : num_points
        base_y(j) = cox_de_boor(i, order, t, steps(j), order);
    end
    ordinal = iptnum2ordinal(i);
    plot(steps, base_y, 'linewidth', 2, '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

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

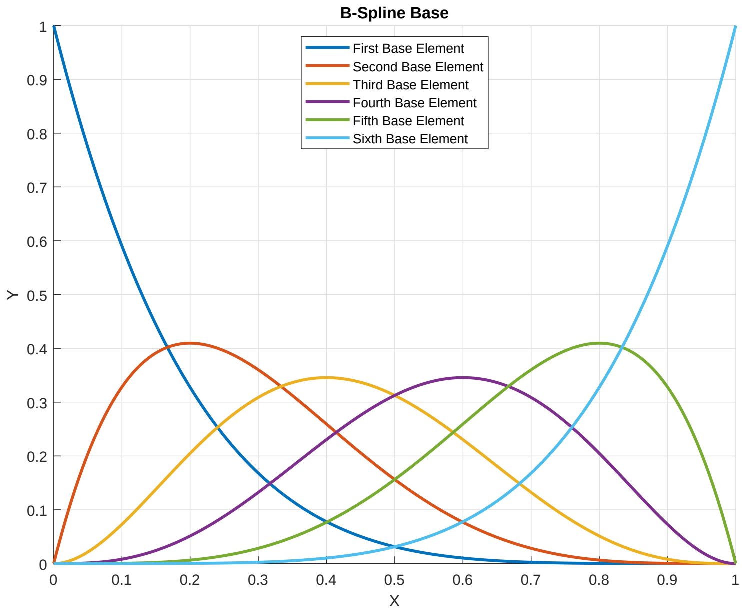
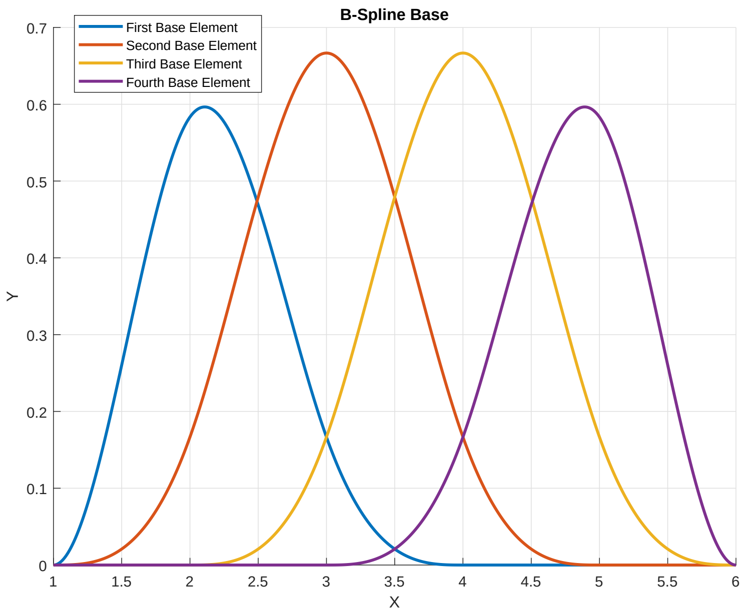


Figura 2: Esempio di base con t = [1, 1, 2, 3, 4, 5, 6, 6] e k= 4



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

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 $\mathbf{u}$ 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.
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.

Listing 3: Algoritmo di De Boor

```
function [curve_point] = de_boor_algorithm(t, t_star, degree, control_points)

    % Find index of knot interval that contains t_star.
    k = find(t <= t_star);
    k = k(end);

    % Calculate the multiplicity of pg t_star in t (0 <= s <= degree+1).
    s = sum(eq(t_star, t));

    % Num. times t_star must be repeated to reach a multiplicity
    % equal to the degree
    h = degree - s;

    % Copy of influenced control points.
    P_ir = zeros(degree+1, size(control_points, 2), degree+1);
    P_ir((k-degree):(k-s), :, 1) = control_points((k-degree):(k-s), :);

    % Main De Boor algorithm.
    q = k - 1;
```

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 $\mathbf{P}_{l-s,degree-s}$

```
    if h > 0
        for r = 1:h
            for i = q-degree+r : q-s
                a_ir = (t_star - t(i+1)) / (t(i+degree-r+2)-t(i+1)
                    );
                P_ir(i+1, :, r+1) = (1 - a_ir)*P_ir(i, :, r) +
                    a_ir * ...
                                        P_ir(i+1, :, r);
            end
        end
        curve_point = P_ir(k-s, :, h+1);
    elseif k == numel(t)
        curve_point = control_points(end, :);
    else
        curve_point = control_points(k-degree, :);
    end
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

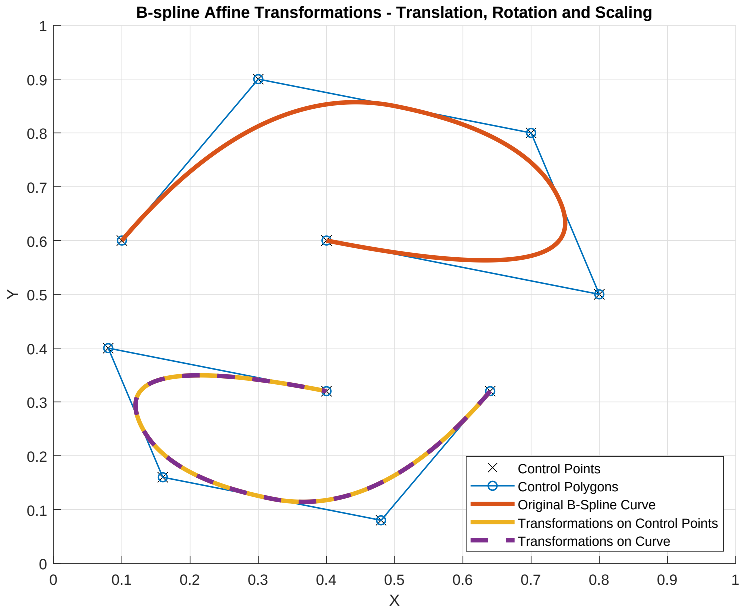
```
clear

% Inputs
left_limit_x = 0;
right_limit_x = 1;
left_limit_y = 0;
right_limit_y = 1;
num_points = 1000;
```

```
t = [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];
num_cp = size(control_points, 1);

% 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([left_limit_x right_limit_x]);
ylim([left_limit_y right_limit_y]);

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

% Plot control points and control polygon.
poi_plot = plot(control_points(:, 1), control_points(:, 2), 'kx', ...
                'MarkerSize', 10);
pol_plot = plot(control_points(:, 1), control_points(:, 2), '-o', ...
                'linewidth', 1, 'color', '#0072BD');

% Calculate and plot the original B-Spline curve using De Boor algorithm.
curve = zeros(num_points, 2);
for i = 1 : num_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
end
original_curve = curve;
original_curve_plot = plot(curve(:, 1), curve(:, 2), 'linewidth', 3, ...
                           'color', '#D95319');

% Tranlation, rotation and scaling transformations.
```

```
T = [0.9 1];
R = [cos(pi) -sin(pi); sin(pi) cos(pi)];
S = [0.8 0; 0 0.8];

% Transformation on control points.
control_points = (control_points*R + T)*S;

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

% Calculate and plot the transformed B-Spline curve.
for i = 1 : num_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
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*R + T)*S;
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.

Figura 3: Trasformazioni Affini di una Curva B-spline



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

```
clear

% Inputs
left_limit_x = 0;
right_limit_x = 1;
left_limit_y = 0;
right_limit_y = 1;
num_points = 1000;
t = [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];
num_cp = size(control_points, 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([left_limit_x right_limit_x]);
ylim([left_limit_y right_limit_y]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(t(degree+1), t(end-degree), num_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_points, 2);
for i = 1 : num_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
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_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
end
plot(curve(:, 1), curve(:, 2), '--', 'linewidth', 3, 'color', '#77AC30');

% Draw lines between modifications.
left_line = de_boor_algorithm(t, t(control_point_mod), degree, ...
                              control_points);
right_line = de_boor_algorithm(t, t(control_point_mod+degree+1), ...
                              degree, control_points);
plot([left_line(1) left_line(1)], [left_limit_y right_limit_y], 'k', ...
    'linewidth', 2)
plot([right_line(1) right_line(1)], [left_limit_y right_limit_y], '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

```
clear

% Inputs
left_limit_x = 0;
right_limit_x = 1;
left_limit_y = 0;
right_limit_y = 1;
num_points = 1000;
t = [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];
num_cp = size(control_points, 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([left_limit_x right_limit_x]);
ylim([left_limit_y right_limit_y]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(t(degree+1), t(end-degree), num_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_points, 2);
for i = 1 : num_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
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_points
    curve(i, :) = de_boor_algorithm(t, steps(i), degree, control_points);
end
plot(curve(:, 1), curve(:, 2), '--', 'linewidth', 3, 'color', '#77AC30');

% Draw lines between modifications.
left_line = de_boor_algorithm(t, t(r), degree, control_points);
right_line= de_boor_algorithm(t, t(r+1), degree, control_points);
plot([left_line(1) left_line(1)], [left_limit_y right_limit_y], 'k', ...
    'linewidth', 2)
plot([right_line(1) right_line(1)], [left_limit_y right_limit_y], '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

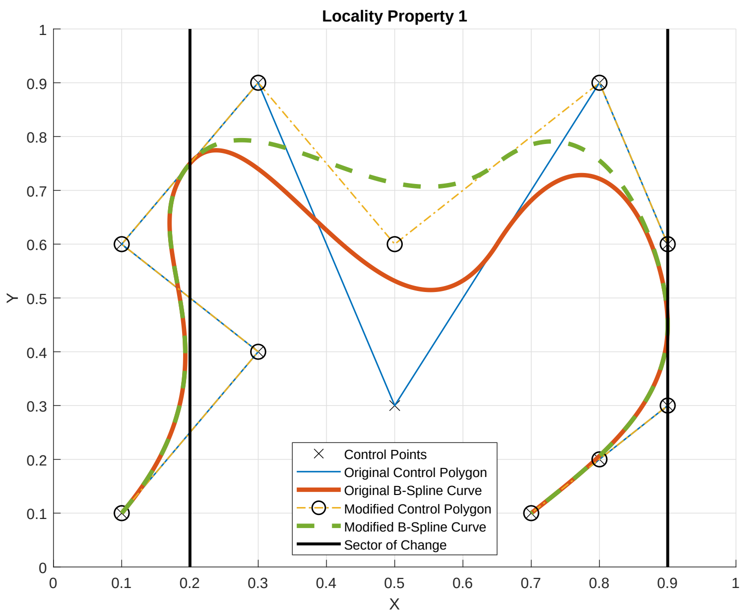
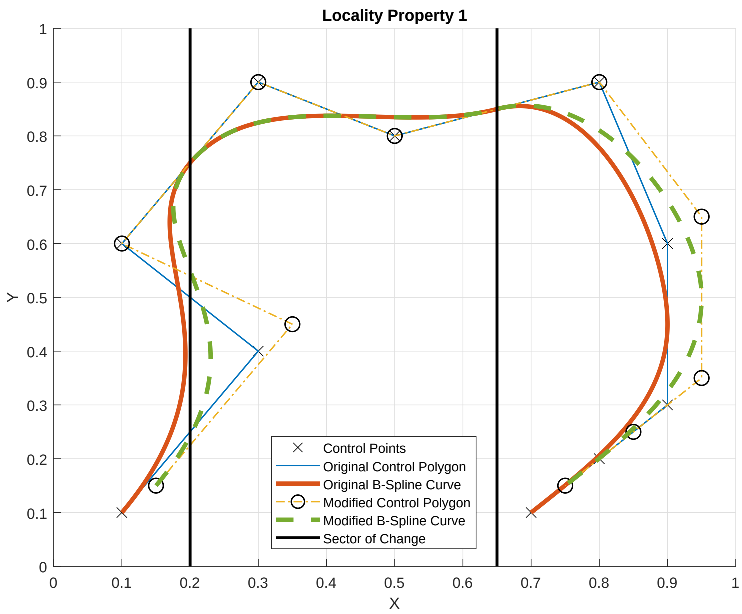


Figura 5: B-spline Località 2



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

Listing 7: Variation diminishing

```
clear

```

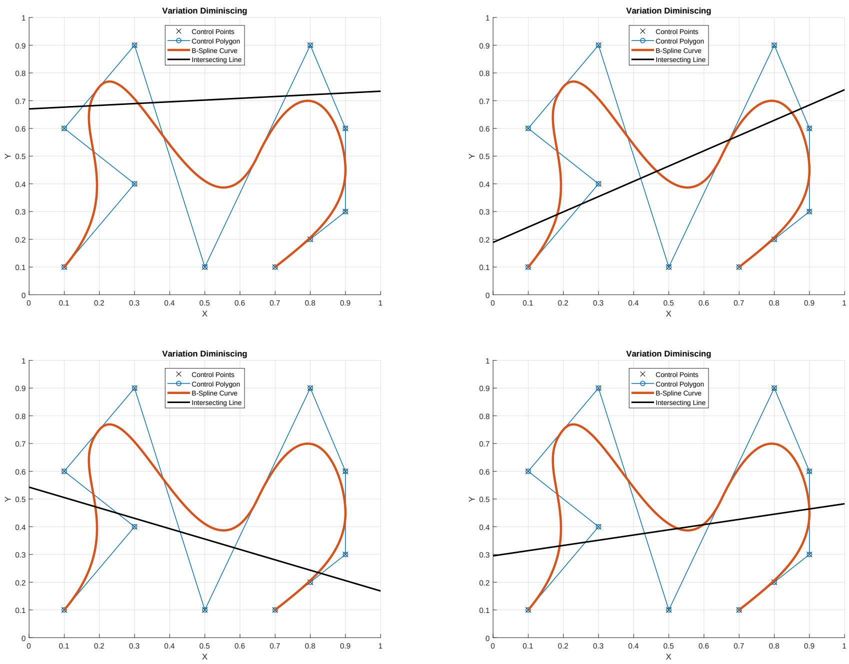


Figura 6: Variation diminishing, quattro esempi di rette

```
% Inputs
num_points = 1000;
t = [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];
num_cp = size(control_points, 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]);

% Calculate the parameter (t) steps for drawing the B-Spline curves.
steps = linspace(t(degree+1), t(end-degree), num_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), '-o', 'linewidth', 1, ...
    'color', '#0072BD');

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

```
clear

prompt = {'Left Limit X Axis:', 'Right Limit X Axis:', ...
          'Left Limit Y Axis:', 'Right Limit Y Axis:', ...
          ['Numbero of Control Points (Last one automatically' ...
          ' generated, V_1=V_n):'], 'Degree:', ...
          'Number of points of the B-Spline curves to draw:'};
dlgtitle = 'Inputs to Draw the B-Spline Curve';
dims = [1 56];
definput = {'0', '1', '0', '1', '5', '2', '1000'};
inputs = inputdlg(prompt, dlgtitle, dims, definput);
left_limit_x = str2double(inputs{1});
right_limit_x = str2double(inputs{2});
left_limit_y = str2double(inputs{3});
right_limit_y = str2double(inputs{4});
num_cp = str2double(inputs{5});
degree = str2double(inputs{6});
num_points = str2double(inputs{7});

% 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([left_limit_x right_limit_x]);
ylim([left_limit_y right_limit_y]);

% Ask user to choose control vertices for the Bezier curve and
    plot them.
control_points = zeros(num_cp + degree + 2, 2);
for i = 1 : num_cp + 1
    if i > num_cp
        % Generate last control point V_1=V_n.
        control_points(num_cp + 1, :) = control_points(1, :);
    else
        [x, y] = ginput(1);
        control_points(i, :) = [x, y];
    end
    poi_plot = plot(control_points(i, 1), control_points(i, 2), '
       kx',....
```

```
            'MarkerSize', 10);
    if i > 1
        pol_plot = plot(control_points(i-1:i,1), control_points(i-1:i, ...
                            2), '-o', 'linewidth', 1, 'color', '#0072BD');
    end
end

% Generate knot vector.
t = -(degree+1)/num_cp : 1/num_cp : (degree+1+num_cp)/num_cp;
control_points(end, :) = control_points(1, :);

% Generate last k (order) control points.
control_points(num_cp+2: end, :) = control_points(2:degree+2, :);
new_poi_plot = plot(control_points(num_cp+2: end, 1), ...
                    control_points(num_cp+2: end, 2), 'g.', ...
                    'MarkerSize', 20);

% Calculate the parameter (t) steps for drawing the Bezier curves.
steps = linspace(t(degree+1), t(end-degree), num_points);

% Calculate and plot the B-Spline curve using De Boor algorithm.
b_spline_curve = zeros(num_points, 2);
for i = 1 : num_points
    b_spline_curve(i, :) = de_boor_algorithm(t, steps(i), degree, ...
                                             control_points);
end
curve_plot = plot(b_spline_curve(:, 1), b_spline_curve(:, 2), ...
                  'linewidth', 3, 'color', '#D95319');
legend([poi_plot pol_plot new_poi_plot curve_plot], {'Control Points', ...
        'Control Polygon', 'New Added Control Points', ...
        'B-Spline Curve'}, 'Location', 'best');
```

Figura 7: Esempio di B-spline Chiusa