UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Elaborato di **Calcolo Numerico**

NURBS e B-Spline - Algoritmi di Knot Insertion e DeBoor

Anno Accademico 2021

Docente: Luisa D'Amore

Aniello Ambrosio	**Mat-0001/19618**
Stefano Aramu	**Mat-0001/19650**
Mario D'Angelo	**Mat-M6300/1242**

Indice

Introduzione

Le B-Spline e le NURBS sono evoluzioni delle curve di Bezier nate agli inizi degli anni 60’ come conseguenza alla necessità di creare un modello matematico che automatizzasse il processo industriale di progettazione e produzione delle automobili.

In particolare, il problema riguardava la gestione automatica delle catene di montaggio per tranciare i pezzi di lamiera, al fine di creare la carrozzeria, seguendo delle traiettorie ben precise.

Dunque, il problema matematico che doveva essere risolto riguardava un disegno di curve 2D/3D che rappresentassero i profili di un automobile.

A tale scopo nacquero dei modelli di rappresentazioni dei dati che si sono evoluti nel tempo giungendo infine alle NURBS e alle B-Spline che ad oggi sono ancora utilizzate per la modellazione di curve.

Capitolo 1: B-Spline e NURBS

In matematica il nome spline denota una funzione costruita da un insieme di polinomi collegati tra loro in modo tale da interpolare un insieme di punti chiamati nodi, garantendo la continuità della funzione fino a un dato ordine di derivate in ogni punto dell'intervallo.

Queste trovano forte applicazione nella grafica computerizzata e nei CAD, presentano svariati vantaggi per la modellazione di oggetti reali tra cui la smoothness, ovvero la possibilità di disegnare una curva che non presenti angoli, necessitano di poco spazio per lo storage in memoria e possono essere modificate in maniera agile tramite i punti di controllo.

Il nostro obbiettivo è quello di trattare le NURBS (NON Uniform Rational B-Splines), ovvero una generalizzazione delle B-Splines che consente di modificare facilmente una curva tramite l'impiego di alcuni pesi associati ai punti di controllo o tramite vettori dei nodi.

Le NURBS trovano largo utilizzo in quanto oltre che risultare semplici nella rappresentazione di forme analitiche standard, risultano flessibili grazi ai punti di controllo e ai pesi, sono efficienti a livello computazionale e invarianti rispetto a operazioni quali cambiamento di scala, traslazione e rotazione che basta applicare sui punti di controllo

Per quanto riguarda però la parametrizzazione di alcune curve "tradizionali" come ad esempio un cerchio, richiede almeno 7 punti di controllo e 10 nodi, mentre in una parametrizzazione normale è necessario conoscere solo raggio e centro.

Prima di introdurre le NURBS, dovremo però fare una breve trattazione riguardante le B-Spline, di cui le NURBS costituiscono una generalizzazione ottenuta tramite pesi.

1.1 B-Spline

Una B-Spline è una curva ottenuta a partire da una combinazione lineare di punti di controllo e funzioni di base, rappresentate nella forma:

$$C(u) = \sum_{i=0}^{n} N_{i,h}(u) P_i$$

Dove $N_{i,h}$ è la funzione di base dell'i-esimo punto di controllo con grado h.

A differenza delle funzioni di base di Bézier quelle B-Spline hanno due importanti proprietà:

- il dominio è suddiviso in nodi
- le funzioni di base non sono nulle nell'intervallo.

Per definire una B-Spline avremo dunque bisogno di alcuni parametri:

- n+1 punti di controllo $P_0,\ldots,P_n$;
- il grado h;
- il vettore dei nodi $T=(u_0,u_1,\ldots,u_{n+h+1})$, con $u_0 \leq u_2 \leq \cdots \leq u_{n+h+1}$

In generale la curva B-Spline non risulta essere chiusa a meno che l'ultimo punto di controllo P_n non coincida con il primo P_0.

La funzione di base va intesa come una funzione di peso che definisce la relazione di influenza tra l'i-esimo punto di controllo P e la curva. Per definire la funzione di base, viene utilizzata una formula ricorsiva :

$$N_{i,0}(u) = \begin{cases} 1 & se\ u_i \leq u \leq u_{i+1} \\ 0 & altrimenti \end{cases}$$

Per le i-esime B-Spline basis function con h=0.

Altrimenti :

$$N_{i,h}(u) = \frac{u - u_i}{u_{i+h} - u_i} N_{i,h-1}(u) + \frac{u_{i+h+1} - u}{u_{i++1} - u_{i+1}} N_{i+1,h-1}(u)$$

La formula appena vista prende il nome di *formula di ricorsione di Cox-DeBoor*.

Le B-Spline godono di alcune proprietà fondamentali :

- Una curva B-Spline sarà divisa in n+p+2 sotto-intervalli.
- Una curva B-Spline è detta *uniforme* se i nodi che la costituiscono sono tra loro equidistanti.

- Se un nodo ha molteplicità p+1 allora è certo che la curva passi per il punto di

controllo corrispondente

- La variazione di un punto di controllo influenza la curva solo nell'intervallo che va da i (indice) a i+p+1, ovvero negli effettivi punti in cui è definita la funzione di base. Nel caso in cui la funzione di base non appartiene all'intervallo, il prodotto della sommatoria per quell'indice restituirà 0 non apportando nessun contributo significativo. (*Proprietà del controllo locale*).

1.2 NURBS

Le NURBS *(Non Uniform Rational B-Spline)* sono quindi una versione pesata delle B-Splines che permettono di rappresentare forme analitiche come le circonferenze in maniera precisa. Se l'equazione di una B-Spline risulta:

$$C(u) = \sum_{i=0}^{n} N_{i,h}(u) P_i$$

Scriveremo quella delle NURBS come

$$C(u) = \sum_{i=0}^{n} P_i \, R_{i,h}(u), \;\; u \in [i, u_{i+h+1}]$$

Dove $R_{i,h}(u)$ sono le funzioni B-Spline razionali e pesate con grado h e vettore dei nodi $T = [u_0, u_1, \dots, u_m]$ e con pesi $w_0, \;\dots, w_n$.

Quindi $R_{i,h}(u)$ può essere scritto come:

$$R_{i,h}(u) = \frac{w_i B_{i,p}(u)}{\sum_{i=0}^{n} w_i B_{i,p}(u)}, \;\; u \in [u_i, u_{i+h+1}]$$

Questa formula può essere facilmente ottenuta a partire da una curva B-Spline, rappresentando ogni punto di controllo in coordinate omogenee $P_i = [x_i \quad y_i \quad z_i \quad 1]^T$.Le coordinate omogenee servono per estendere quelle euclidee con il concetto di punto all'infinito, ovvero in uno spazio monodimensionale considerano una coppia di valori (a, w) come punto sulla retta a/w , se w risulta diverso da zero, il valore del punto sarà a/w altrimenti divergerà a ∞.

Per passare dalle coordinate euclidee a quelle omogenee bisogna rappresentare (x, y) come $(x \cdot w, y \cdot w, w)$, mentre per attuare la trasformazione inversa, basta dividere le prime due coordinate per la terza.

Estendendo il discorso a quello di una funzione $f(x, y) = 0$, rimpiazzando x, y con $(x/w, y/w)$ per ogni punto, otterremo $f(x/w, y/w) = 0$.

Se la funzione risulta essere polinomiale, il prodotto con w^h con h grado del polinomio, tutti i denominatori diventeranno pari a 1 e tutti i termini risulteranno avere lo stesso grado del polinomio d'origine.

La divisione per w^h e la sostituzione di $x/w^h \;\; e \;\; y/w^h$ con $x \, e \, y$ permetterà la conversione della funzione in coordinate euclidee. La trasformazione da coordinate omogenee vede un punto in due dimensioni, come un punto di tre dimensioni proiettato in un piano dove w=1. Ritornando al discorso dei punti di controllo, potremo quindi

rappresentare ogni punto in coordinate omogenee

$$P_i = [x_i \quad y_i \quad z_i \quad 1]^T$$

La moltiplicazione per uno scalare di tutte le coordinate non impatterà su quelle euclidee.

Possiamo quindi moltiplicare le nostre coordinate per il peso w_i .

$$P_i^w = [w_i x_i \quad w_i y_i \quad w_i z_i \quad w_i]^T$$

Esprimiamo dunque la B-Spline in coordinate omogenee:

$$C^w(u) = \sum_{i=0}^{n} N_{i,h}(u) P_i^w = \sum_{i=0}^{n} N_{i,h}(u) \begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} N_{i,h}(u)(w_i x_i) \\ \sum_{i=0}^{n} N_{i,h}(u)(w_i y_i) \\ \sum_{i=0}^{n} N_{i,h}(u)(w_i z_i) \\ \sum_{i=0}^{n} N_{i,h}(u) w_i \end{bmatrix}$$

Dividendo per la quarta coordinata effettuiamo la trasformazione in coordinate euclidee ottenendo l'effettiva rappresentazione della NURBS:

$$\mathrm{C}(u) = \begin{bmatrix} \frac{\sum_{i=0}^{n} N_{i,h}(u)(w_i x_i)}{\sum_{i=0}^{n} N_{i,h}(u) w_i} \\ \frac{\sum_{i=0}^{n} N_{i,h}(u)(w_i y_i)}{\sum_{i=0}^{n} N_{i,h}(u) w_i} \\ \frac{\sum_{i=0}^{n} N_{i,h}(u)(w_i z_i)}{\sum_{i=0}^{n} N_{i,h}(u) w_i} \\ 1 \end{bmatrix} = \sum_{i=0}^{n} \frac{N_{i,h}(u) w_i}{\sum_{j=0}^{n} N_{j,h}(u) w_j} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \frac{\sum_{i=0}^{n} w_i P_i N_{i,h}(u)}{\sum_{i=0}^{n} w_i N_{i,h}(u)}$$

Possiamo quindi affermare che la NURBS può essere interpretata come la proiezione di una B-Spline in uno spazio tetradimensionale (4D).

Le NURBS oltre a godere delle proprietà delle B-Spline (essendo una generalizzazione di quest'ultime), godono di una serie di proprietà molto importanti:

- *Generalizzazione:* se tutti i w=1→ $Ri, h(u) = \begin{cases} Bi, h(u) \; se \;\; T = [0, \dots, 0, \; 1, \; \dots, \; 1) \\ Ni, \; h(u) \; altrimenti \end{cases}$ con $Bi, p(u)$ polinomi Bernstein
- *Partizione dell'unità:* $\sum_{k=0}^{n} Ri, h(u) = 1$
- *Località:* $Ri, h(u) = 0 \;\; se \; u$ non appartiene a $[ui, \; ui + h + 1]$
- *Controllo locale:* se un punto di controllo viene modificato, questo impatta la curva solo negli h+1 intervalli.
- *Convex Hull:* la curva è totalmente contenuta in un poligono convesso dato dai punti di controllo. Risulta anche che per $u \; che \; appartiene \; [ui, \; ui + 1] \; C(u)$ si trova all'interno di questo poligono convesso.
- *Differenziabilità:* le funzioni di base sono continue e differenziabili h-k volte dove k è la molteplicità

- *Invarianza alle trasformazioni affini prospettiche:* le curve risultano invarianti a questo tipo di trasformazioni e queste ultime possono essere applicate direttamente a punti di controllo.

1.2.1 Pesi

Denotando con w un peso, esso influenza la curva solo nell'intervallo $[ui,\ ui+1+1)$. Supponendo che vari unicamente il peso, definiamo i punti :

$$\boldsymbol{B} = \boldsymbol{C}(u;\ w_i = 0)$$

$$\boldsymbol{N} = \boldsymbol{C}(u;\ w_i = 1)$$

$$\boldsymbol{B_i} = \boldsymbol{C}(u; w_i \neq 0\ [0,1])$$

Con i parametri

$$\alpha = R_{i,h}(u;\ w_i = 1)$$

$$\beta = R_{i,h}(u)$$

Possiamo riscrivere N e Bi come:

$$\boldsymbol{N} = (1-\alpha)\boldsymbol{B} + \alpha \boldsymbol{P}_i$$

$$\boldsymbol{B_i} = (1-\beta)\boldsymbol{B} + \beta \boldsymbol{P}_i$$

Con α e β otteniamo l'identità:

$$\frac{1-\alpha}{\alpha} : \frac{1-\beta}{\beta} = \frac{\boldsymbol{P_iN}}{\boldsymbol{BN}} : \frac{\boldsymbol{P_iB_i}}{\boldsymbol{BB_i}} = w_i$$

Con le equazioni appena scritte possiamo analizzare quali siano gli effetti della modifica dei pesi sulla forma della curva:

- Al crescere di $w_i \rightarrow \beta$ cresce e la curva si avvicina al punto P_i.
- Al decrescere di $w_i \rightarrow \beta$ decresce e quindi la curva si allontana da P_i.
- Se w_i cresce, la curva viene allontanata da P_j con j diverso da i.
- Se w_i decresce allora la curva viene avvicinata a P_j con j diverso da i.

- Man mano che B_i tende a P_i, β si avvicina a 1 e w_i tende a divergere.

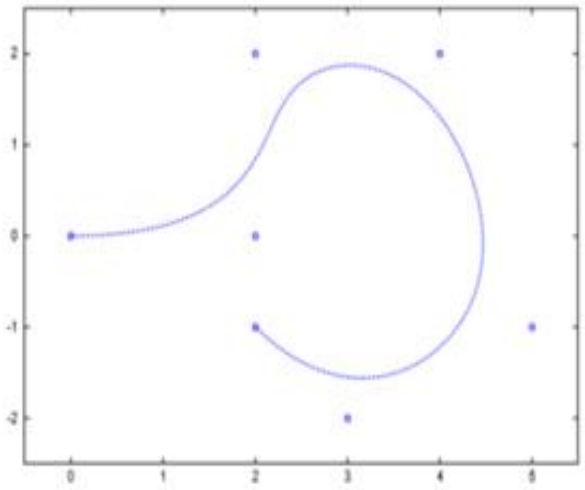


Figure 2: Punti di controllo

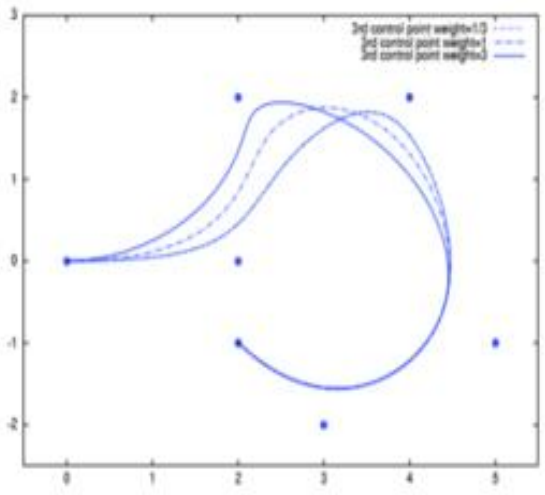


Figure 3: Curva NURBS con variazione del peso del terzo punto di controllo

Capitolo 2: Algoritmo di DeBoor e Knot Insertion

2.1 Algoritmo di DeBoor

Oltre all'algoritmo di *Cox-DeBoor* per il calcolo delle B-Spline, esteso poi anche al calcolo delle NURBS, viene utilizzato l'algoritmo di *DeBoor* che permette di calcolare in maniera più rapida i punti in cui passerà la B-Spline o la NURBS.

Per valutare il valore della curva nel punto, l'algoritmo si basa sull'inserimento del nodo u $\in [u_k , u_{k+1}[$, dove k è l'indice di intervallo a cui appartiene il nodo, come nodo di molteplicità h pari al grado della curva. L'ultimo inserimento del nodo rappresenterà il punto di controllo per la quale passerà la curva.

Ricordiamo che quanto più aumenta la molteplicità di un nodo, tanto più si riduce il numero di funzioni di base non nulle sul nodo, e se questa raggiunge proprio il valore h, solo una delle funzioni di base avrà valore non nullo.

L'Algoritmo di DeBoor sfrutta la *Knot Insertion* per l'inserimento ripetuto del nodo e per il calcolo del nuovo punto di controllo.

Il funzionamento dell'algoritmo è molto semplice:

- Se la molteplicità del nodo u è 0, allora il nodo va inserito un numero di volte uguale al grado della curva, altrimenti va inserito un numero di volte pari alla differenza tra il grado della curva e la molteplicità del punto. Più in generale sia *mol* la molteplicità del nodo u e *h* il grado della curva allora:

$$numero\_inserimenti = h - mol$$

Per l'inserimento del nodo viene sfruttato l'algoritmo di Knot Insertion facendo però alcune premesse:

- Si suppone che il nodo da inserire u $\in [u_k , u_{k+1}[$.
- Si suppone che i punti di controllo che influenzino il nodo u siano al più h-mol+1 e vadano da $Punti\_Controllo_{k-h}$ a $Punti\_Controllo_{k-mol}$.
- Si suppone che i punti inseriti dalla Knot Insertion siano h-mol (non h-mol+1).

Al termine dell'algoritmo verrà restituito un punto che sarà proprio quello per la quale passerà il punto.

Come precedentemente accennato, l'algoritmo può essere adoperato sia per il calcolo delle B-Spline che delle NURBS. Nel caso delle NURBS, prima di applicare l'algoritmo bisognerà portare il vettore dei punti di controllo in tre dimensioni. A tale scopo si utilizza il vettore dei pesi:

Ciascun punto di controllo verrà moltiplicato per il relativo peso che poi verrà aggiunto come terzo parametro del vettore. Dopo aver applicato l'algoritmo di DeBoor sul vettore così calcolato basterà riproiettare il vettore risultante in due dimensioni.

2.1.1 IMPLEMENTAZIONE IN MATLAB

```
function [C]=Algoritmo_DeBoor(Punti_C,num_nodi,t,h)
%-------------------------------------------------------------------------
% DESCRIZIONE
%-------------------------------------------------------------------------
% L'algoritmo è utilizzato come alternativa più rapida per il calcolo delle
% B-Spline e delle NURBS.
% L'idea alla base dell'algoritmo è quella di andare ad inserire più volte
% uno stesso nodo u per fare in modo che la sua molteplicità raggiunga il
% valore h, ovvero il grado della curva.
% L'algoritmo fa uso dell' algoritmo di knot insertion per inserire
% ripetutamente il nodo all'interno di un vettore.
% Supponendo che il nodo da inserire sia contenuto in [uk, uk+1),i punti
% di controllo che lo influenzano sono h — mol + 1, e in particolare vanno
% da Pk—h a Pk—mol mentre i punti calcolati nell'iterazione della
% Knot Insertion sono h — mol, diminuendo quindi di 1 a causa dell'aumento
% di molteplicità di u.
% L'algoritmo termina quindi quando avremo un solo punto di controllo
% risultante, che è proprio il punto per il quale passa la curva.
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% L'algoritmo prende in input:
%   - La matrice Punti_C dei punti di controllo
%   - Il numero di nodi num_nodi
%   - Il vettore dei nodi t
%   - Il grado della funzione h
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% L'algoritmo restituisce in uscita la matrice di punti C in cui passerà il
% grafico.
%-------------------------------------------------------------------------
% vettore di punti in cui viene calcolata la curva
Punti_Calcolo= linspace(t(num_nodi), t(end-h), 10*size(Punti_C,2));
%calcolo del numero di righe della matrice da plottare
m = size(Punti_C,1);
%Pk contiene un numero di matrici pari all'ordine della curva.
Pk = zeros(m, num_nodi, num_nodi); % m righe, num_nodi colonne, num_nodi dimensioni

coefficienti = zeros(num_nodi, num_nodi);
 for i=1: size(Punti_Calcolo,2) %Algoritmo da seguire per ogni punto di calcolo per ↙
plottare la B-spline
     coefficienti(:)=0;
     Pk(:)=0; %reset per ogni iterazione delle matrici Pk.
     p_c= Punti_Calcolo(i);
     mol=0; %reset moltepicità ad ogni iterazione
%      Calcolo della molteplicità dei punti di calcolo:
%      sereve a determinare numero di Knot Insertion che devono essere effettuate.
     for j=1: size(t,2)
         if p_c==t(j)%se riscontra una corrispondenza con un nodo
             mol=mol+1;%incrementa la molteplicità del punto di calcolo
         end
     end
```

```
%       Adesso si calcola l'intervallo giusto in cui andare ad inserire il
%       punto di valutazione.
%       Si calcola quindi i'indice k : u appartiene all'intervallo [tk, tk+1]
%       A tale scopo utilizziamo la funzione histc:
%       La funzione histcounts permette di calcolare i'intervallo del vettore
%       dei nodi in cui andare ad inserire il punto di calcolo p_c.

      [~,k]=histc(p_c,t);

      num_inserimenti= h-mol;
%      Il punto di controllo viene inserito tante volte quanto la
%      differenta tra il grado della curva e la molteplicità del punto

%      A causa della proprietà del controllo locale, i punti di controllo che
%      vengono influenzati dall'algoritmo non sono tutti, ma in generale sono
%      k-h (indice dell'intervallo - il grado della curva). Considerando
%      anche la molteplicità del punto di valutazione che stiamo inserendo, i
%      punti di controllo influenzati sono quelli che vanno da k-h fino a
%      k-mol.

      Pk(:,(k-h):(k-mol),1) = Punti_C(:,(k-h):(k-mol));

%       Se num_inserimenti >0, allora vi saranno inserimenti da effettuare.
%       Altrimenti si procede secondo due operazioni diverse possibili:
%       1)il nodo va inserito nell'ultimo intervallo;
%       2)la molteplicità del nodo è pari al grado della curva e quindi
%       già ne conosciamo il valore e lo inseriamo nel vettore di calcolo.

      if num_inserimenti>0
              for z=1:num_inserimenti
%si utilizza la knot insertion per inserire ripetutamente il nodo
                Pk=InserimentoNodi(Pk,h,k,mol,z,p_c,t,coefficienti);
              end
              %il punto viene inserito nel vettore da plottare
              C(:,i) = Pk(:,k-mol,h-mol+1);
      elseif  k==size(t,2)
%inserimento del punto di controllo nell'ultimo intervallo
           C(:,i) = Punti_C(:,end);
      else
%valore noto aggiunto al vettore di calcolo.
          C(:,i) = Punti_C(:,k-h);
      end
 end
end
```

2.1.2 *IMPLEMENTAZIONE IN MATLAB PLOT B-SPLINE TRAMITE DEBOOR*

```
function DeBoor (Punti_C,h,t,handles)

%-------------------------------------------------------------------------
% DESCRIZIONE FUNZIONE
%-------------------------------------------------------------------------
% L'algoritmo di DeBoor permette di calcolare la curva B-Spline.
%
% Riassunto funzionamento:
%
% La curva viene calcolata su un insieme di punti contenuti in un vettore
% generato all'interno della funzione stessa e i cui valori saranno
% interni al range dei nodi.
% Per ognuno di questi punti :
%   -  viene valutata la molteplicità;
%   -  viene trovato il giusto intervallo in cui inserire il punto;
%   -  tramite la funzione Knot_Isertion il punto viene inserito
%      iterativamente finchè la molteplicità del punto di valutazione
%      risulta uguale al grado della curva.
%      Se la molteplicità del punto di valutazione è originariamente uguale
%      al grado della curva non ci sarà alcun inserimento.
%
% Casi di Errore:
%
%   - i nodi inseriti in t non reali;
%   - numero nodi inseriti in t non corretto;
%   - vettore dei nodi t non ordinato;
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% La funzione ha in input:
%   - la matrice Punti_C contenente le coordinate dei punti di controllo.
%   - Grado della curva h
%   - Vettore dei nodi t
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% La funzione restituisce in output il plot della B-Spline.
%-------------------------------------------------------------------------



axis on;
grid on;
hold on;

num_nodi= h+1; %grado della curva bspline
%CONTROLLO ERRORI
num_nodi_corretto= size(Punti_C,2)+h+1; % num_nodi è il numero di nodi da inserire

% verifica che i nodi inseriti siano dei numeri reali
set(handles.errori, 'String', ' I nodi devono essere dei numeri reali.');
validateattributes(t, {'numeric'}, {'real','vector'});
set(handles.errori, 'String', ' ');
```

```
% verifica che il numero di nodi inseriti sia corretto (uguale a num_nodi_corretto)
if (size(t,2)~=num_nodi_corretto)
    errNodi=sprintf('Il numero di nodi devono essere: %d!',num_nodi_corretto);
    set(handles.errori, 'String', errNodi);
    assert( size(t,2)==num_nodi_corretto);
% verifica che il vettore dei nodi sia ordinato in ordine crescente
elseif (any( t(2:end)-t(1:end-1) < 0))
    set(handles.errori, 'String', 'I nodi devono essere inseriti in ordine ↙
crescente');
    assert(not(any( t(2:end)-t(1:end-1) < 0))); % RIPORTA ERRORE SE I NODI NON SONO ↙
CRESCENTI
end

C=Algoritmo_DeBoor(Punti_C,num_nodi,t,h);
line(C(1,:), C(2,:),'LineWidth',2);                                    %la funzione ↙
line disegna la curva in base al vettore C costruito.
end
```

L'algoritmo prima di procedere al calcolo dei punti tramite su cui tracciare la curva usando il comando line, effettua una serie di controlli per garantire la robustezza del software:

- Effettua un primo controllo verifica che i nodi inseriti siano reali.
- Un secondo controllo verifica che il numero di nodi inserito sia corretto.
- Il terzo controllo effettua una verifica sull'ordinamento crescente del vettore dei nodi.

In caso di fallimento viene generata sulla GUI un messaggio momentaneo di errore, la correzione dell'errore permette la ripresa dell'esecuzione del programma.

Dopo la verifica degli errori, utilizzando la funzione Algoritmo_DeBoor(), che prendendo in input il vettore dei punti di controllo, il numero di nodi, il vettore dei nodi t e il grado della funzione h, restituisce un vettore C di punti in cui tracciare la B-Spline.

La B-Spline viene tracciata tramite il comando line.

2.1.3 *IMPLEMENTAZIONE IN MATLAB PLOT NURBS TRAMITE DEBOOR*

```
function DeBoor_NURBS (Punti_C, h, t, w, handles)

%-------------------------------------------------------------------------
% DESCRIZIONE FUNZIONE
%-------------------------------------------------------------------------
% La funzione implementa l'algoritmo di De Boor adattato alle NURBS.
%
% Riassunto funzionamento:
%
% La principale differenza con rispetto al caso delle B-Spline sta nel
% fatto che per le Nurbs avremo dei pesi per ogni punto di controllo.
% Per poter applicare l'algoritmo di de Boor dovremo quindi convertire
% i punti di controllo 2D ,utilizzati per il tracciamento delle B-Spline,
% in punti di controllo a 3 dimensioni, dove:
%   - le prime due dimensioni sono pari alle coordinate moltiplicate per il
%     peso del punto di controllo stesso
%   - la terza il peso stesso del punto di controllo
% Infine, dopo aver applicato l'algoritmo di DeBoor sui punti di calcolo a
% tre dimensioni, si proietteranno quest'ultimi nuovamente in due
% dimensioni, per il tracciamento delle curve di NURBS.
%
% Casi di Errore:
%
% Oltre ai casi d'errore in cui ci si trovava nel caso di DeBoor applicato
% alle B-Spline, l'algoritmo va in errore se:
%   - il numero dei pesi inserito non è pari al numero dei punti di
%     controllo
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% La funzione ha in input:
%   - la matrice Punti_C contenente le coordinate dei punti di controllo.
%   - Grado della curva h
%   - Vettore dei nodi t
%   - Vettore dei pesi w
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% La funzione restituisce in output il plot della NURBS.
%-------------------------------------------------------------------------

axis on;
grid on;
hold on;

num_nodi= h+1; %grado della curva bspline
num_nodi_corretto= size(Punti_C,2)+h+1; % num_nodi è il numero di nodi da inserire

% verifica che i nodi inseriti siano dei numeri reali
set(handles.errori, 'String', ' I nodi devono essere dei numeri reali.');
validateattributes(t, {'numeric'}, {'real','vector'});
set(handles.errori, 'String', ' ');
```

```
% verifica che il numero di nodi inseriti sia corretto (uguale a num_nodi_corretto)
if ( size(t,2)~=num_nodi_corretto)
    errNodi=sprintf('Il numero di nodi devono essere: %d!',num_nodi_corretto);
    set(handles.errori, 'String', errNodi);
    assert( size(t,2)==num_nodi_corretto);
% verifica che il vettore dei nodi sia ordinato in ordine crescente
elseif (any( t(2:end)-t(1:end-1) < 0))
    set(handles.errori, 'String', 'I nodi devono essere inseriti in ordine ↙
crescente');
    assert(not(any( t(2:end)-t(1:end-1) < 0))); % RIPORTA ERRORE SE I NODI NON SONO ↙
CRESCENTI
elseif (size(w,2)~=size(Punti_C,2))
    set(handles.errori, 'String', 'Il vettore dei pesi deve avere tanti pesi quanti ↙
sono i punti di controllo');
    assert(size(w,2)==size(Punti_C,2));
end

Punti_C = [Punti_C(1,:) .* w(1,:); Punti_C(2,:) .* w(1,:); w];
% aggiusto la matrice dei punti di controllo aggiungendo i pesi e
% moltiplicando questi ultimi alle coordinate corrispondenti.

% vettore di punti in cui viene calcolata la curva
C=Algoritmo_DeBoor(Punti_C,num_nodi,t,h);
% proietto nuovamente i punti di controllo
% in due dimensioni per ottenere la NURBS
Punti_C(1:2,:) = Punti_C(1:2,:)./Punti_C(3,:);
C(1:2,:) = C(1:2,:)./C(3,:);

line(C(1,:), C(2,:), 'LineWidth',2); %con la funzione line viene disegnata la curva
end
```

Anche nel caso del calcolo delle NURBS, l'algoritmo prima di calcolare i punti in cui passa la funzione, effettua un controllo su condizioni che potrebbero dare errori, in particolare oltre ai casi eccezionali visti per le B-Spline, si aggiunge il controllo su un quarto parametro che potrebbe portare in errore:

- Viene effettuato un controllo sul numero di pesi inseriti che deve essere uguale al numero di punti di controllo.

Successivamente al controllo, prima di applicare l'Algoritmo di DeBoor, viene aggiornato il vettore dei punti andando a moltiplicare ogni punto per il relativo peso e aggiungendo un terzo campo peso al punto di controllo.

Applicato DeBoor ci verrà restituito un vettore su cui tracciare la curva che conterrà anche i pesi dei punti presenti al suo interno, per cui prima di utilizzare il comando line si procederà a proiettare nuovamente sia i punti di controllo che i punti del vettore C in due dimensioni.

Da notare l'algoritmo di DeBoor è stato implementato in una funzione autonoma e richiamato nell'applicazione NURBS e B-SPLINE. Questa scelta di progetto è stata adottata per evitare ridondanza di codice e garantire il riuso.

Si è deciso di implementare anche la Knot Insertion in una funzione autonoma InserimentoNodi(), al fine di poter riutilizzare il modulo nel caso in cui si volessero creare future implementazioni per aggiungere Nodi da GUI.

2.2 Knot Insertion

L'algoritmo di Knot Insertion risulta essere di fondamentale importanza sia nel calcolo delle B-Spline che in quello delle NURBS. Esso permette di:

- Valutare punti su curve e superfici tramite l'algoritmo di DeBoor
- Suddividere curve e superfici
- Aggiungere nuovi nodi nel vettore senza cambiare la forma dell'intera curva decretando così un interattività del design.

Partendo dal vincolo *num_nodi=num_p+h+1*, ovvero il numero di nodi deve essere dato dalla somma dei punti di controllo il grado delle funzioni di base +1, se inseriamo un nuovo nodo, deve cambiare anche il numero dei punti di controllo o il grado delle funzioni. Aumentare il grado delle funzioni di base non è però una scelta ottimale in quanto modificandolo si andrà a modificare la curva complessivamente e non localmente, per cui all'inserimento di un nodo è necessario inserire un punto di controllo che lascerà comunque inalterata morfologicamente la nostra curva.

Dati n+1 punti di controllo $[P_0, P_1,\ldots, P_n]$ ed un vettore di m+1 nodi $t=\{u_0, u_1,\ldots, u_m\}$ e il grado *h* della curva, vogliamo inserire un nuovo nodo u nel vettore t. Supponendo che il nuovo nodo da inserire si trovi nello span $[u_k,u_{k+1}]$, per la proprietà del controllo locale sappiamo che la curva, NURBS o B-Spline che sia, sarà interna al convex hull definito dai soli punti di controllo. Per inserire il nuovo nodo quindi c'è bisogno di trovare num_p nuovi punti di controllo, tali punti si troveranno sui segmenti che collegano i punti di controllo originari:

- Q_k si troverà sul segmento $P_{k-1}P_k$
- Q_{k-1} si troverà sul segmento $P_{k-2}P_{k-1}$

- ….
- Q_{k-p+1} si troverà sul segmento $P_{k-p}P_{k-p+1}$

In tal modo si otterrà un nuovo convex hull. L'algoritmo darà molto utile in quanto genererà a partire da num_p-1 punti num_p punti. Il k-esimo punto di controllo Q_k appartenente al segmento $P_{k-1}P_k$ sarà dato dalla formula:

$$Q_k = (1 - a_i)P_{i-1} + a_i P_k$$

Dove il generico coefficiente a_i viene calcolato come:

$$a_i = \frac{u - u_i}{u_{i+h} - u_i}$$

Nel caso in cui nuovo nodo coincida con un nodo già presente nel vettore dei nodi t allora aumenterà anche la molteplicità del punto.

È possibile applicare la Knot Insertion anche alle NURBS sotto alcune condizioni, infatti, ricordando che le NURBS sono una proiezione in 4D delle curve B-Spline in 3D, prima di applicare l'algoritmo bisognerà convertire la NURBS in una B-Spline in 4D, applicare l'algoritmo e infine riproiettare i punti di controllo su 3 dimensioni formando nuovamente un insieme di punti di controllo validi per la NURBS:

I punti di controllo della NURBS saranno $P_i = (x_i, y_i, w_i)$, per la B-Spline in 4D invece $P_i^w = (x_i w_i, y_i w_i, z_i w_i\ , w_i)$, l'inserimento del nuovo nodo u, produrrà quindi punti di controllo $Q_i^w = (x_i, y_i, z_i\ , w_i)$, per riportare il punto in 3D basterà dividerli tutti per la quarta componente.

2.2.1 *IMPLEMENTAZIONE IN MATLAB KNOT INSERTION*

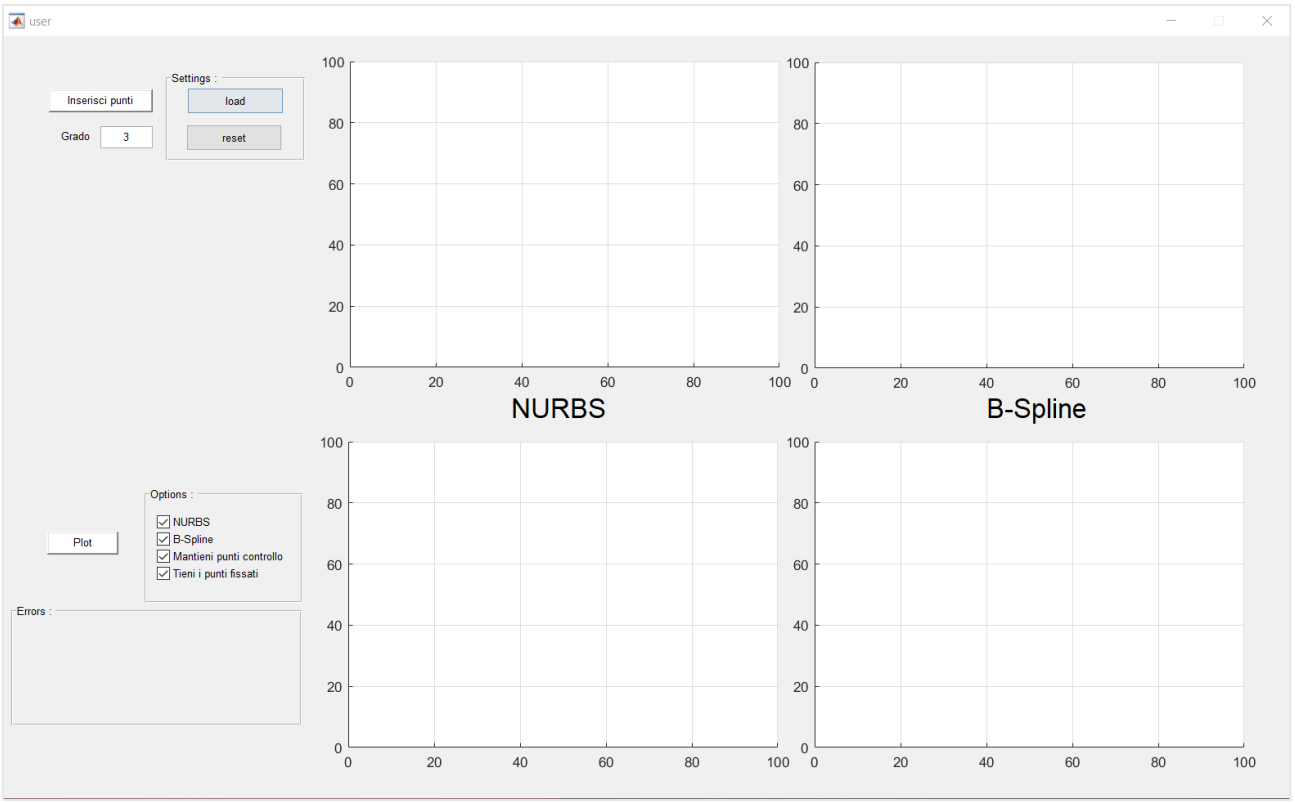
```
function [Pk_new]=InserimentoNodi(Pk,h,k,mol,z,p_c,t,coefficienti)
%-------------------------------------------------------------------------
% DESCRIZIONE FUNZIONE
%-------------------------------------------------------------------------
% La funzione permette di inserire un nodo all'interno del
% vettore di nodi t senza che venga alterata la forma della curva tramite
% l'algoritmo di Knot Insertion.
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% La funzione ha in input:
%   - la matrice Pk contenente le coordinate dei punti di controllo;
%   - il grado h della curva;
%   - l'indice k che indica l'intervallo in cui inserire il nodo;
%   - la molteplicità mol del nodo da inserire;
%   - un indice z per gestire il numero di iterazioni;
%   - il nodo da inserire p_c;
%   - il vettore dei nodi t;
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% La funzione restituisce in output:
% La matrice contente le coordinate dei nuovi punti di controllo Punti_C
%-------------------------------------------------------------------------

b = k-1;                                                                %↙
variabile ausiliaria
for i = (b-h+z) : (b-mol)
    coefficienti(i+1,z+1) = (p_c-t(i+1)) / (t(i+h-z+1+1)-t(i+1)); ↙
% calcolo dei coefficienti
    Pk(:,i+1,z+1) = (1-coefficienti(i+1,z+1)) * Pk(:,i,z) + coefficienti(i+1,z+1) * ↙
Pk(:,i+1,z); % calcolo dei punti di controllo
end
    Pk_new=Pk;
end
```

La funzione implementata in matlab permette di calcolare i nuovi punti di calcolo che poi saranno restituiti all'Algoritmo di DeBoor per il calcolo dei punti in cui passerà la curva. Tuttavia, potrebbe essere utilizzata per altri scopi come, ad esempio, per la creazione di un modulo che permetta di inserire nuovi nodi a partire dalla GUI.

Capitolo 3: GUI e Implementazione

L'interfaccia grafica del programma è stata totalmente realizzato con GUIDE, un tool messo a disposizione da matlab per la creazione di una GUI a cui è possibile interfacciare più script. È possibile lanciare l'interfaccia aprendo ed eseguendo il file user.m presente nella cartella dell'elaborato.

Aperta l'interfaccia si notano subito tre pannelli laterali:



Settings, che contiene i tasti :

- load: utilizzato per selezionare un'immagine dal proprio PC e usarla per la scelta dei punti di controllo.
- Reset: con una callback resetta tutti i valori del programma a 0 e pulisce gli assi.

Options, contenente:

- NURBS: checkbox che se spuntata traccia le curve NURBS alla pressione del tasto "Plot".
- B-Spline: checkbox che se spuntata traccia le curve B-Spline alla pressione del tasto "Plot".

- Mantieni punti di controllo: checkbox che se spuntata considera nella stampa delle curve anche quella dei punti di controllo selezionati dall'utente.
- Tieni i punti fissati: checkbox che se spuntata genera una curva "fissata", altrimenti "aperta".

Errors, contenente il dialog error che gestisce e mostra a video gli errori generati dall'utente nell'utilizzo del programma.

I *buttons* messi a disposizione per l'utente sono:

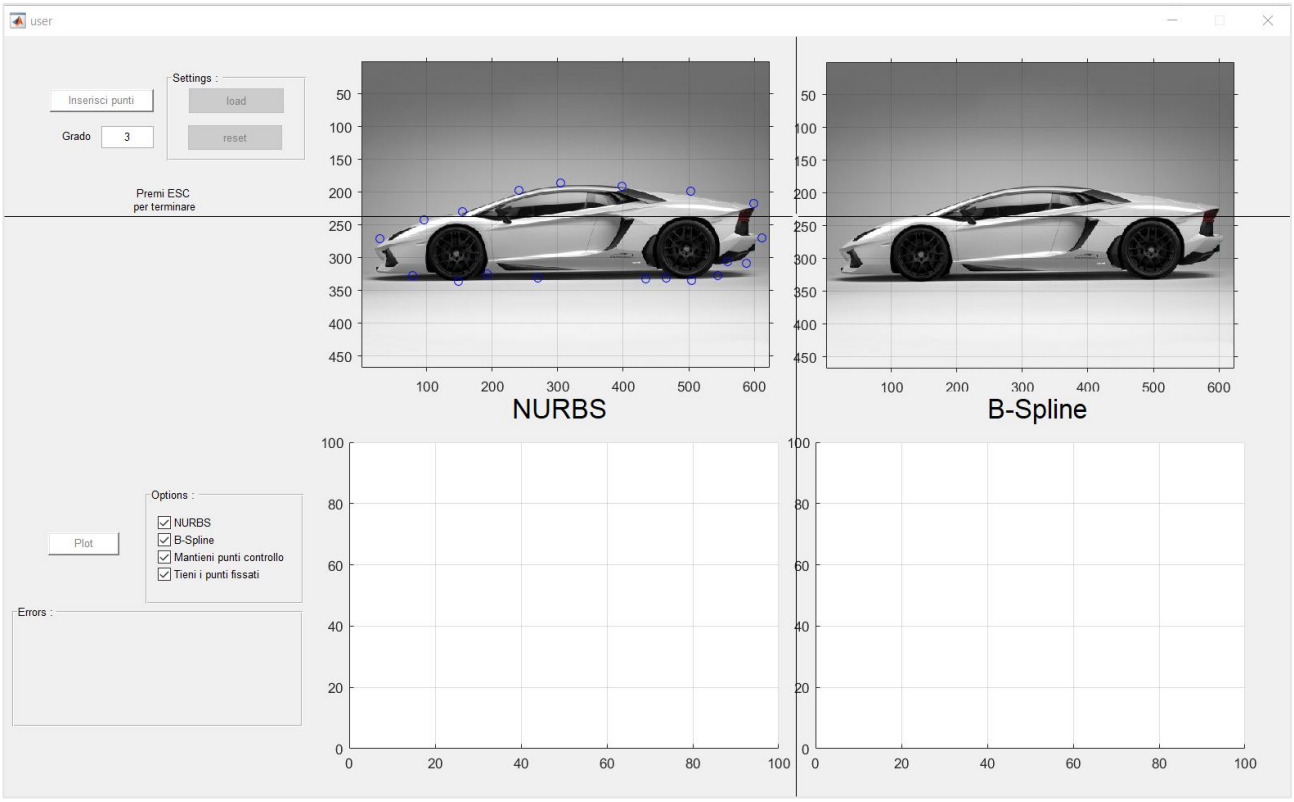
- Inserisci i punti, che tramite callback richiama la funzione Selezione_Punti che permette appunto la selezione dei punti di controllo.
- I già citati load e reset.
- Plot, che permette di stampare sugli assi le curve B-Spline e le NURBS, in base ai valori del pannello Options selezionati.

Inoltre, è presente un campo di testo modificabile "Grado" che permette di scegliere il grado dei polinomi (di default questo campo ha come valore "3").

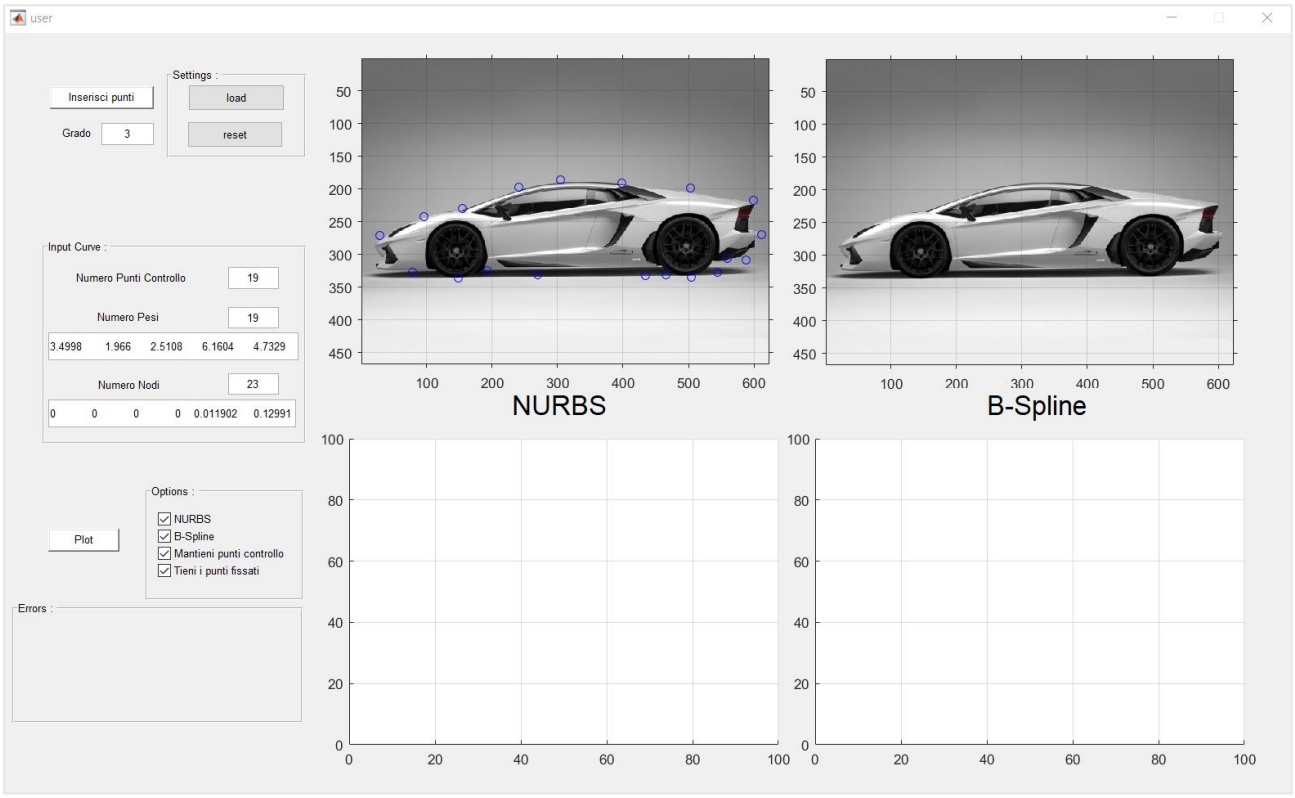
La prima cosa che dovrà fare l'utente per interagire con il programma è il caricamento dell'immagine tramite il tasto load.

In seguito, dovrà essere cliccato il tasto Inserisci punti per iniziare a posizionare i punti di controllo.

Alla pressione del tasto i restanti bottoni verranno disattivati per impedire all'utente di chiamare altre callback. Un testo suggerirà all'utente di premere il tasto ESC per terminare l'inserimento. Soltanto in seguito, in caso di errore nel tracciamento l'utente potrà premere il tasto reset e riavviare la procedura.



Premendo il tasto ESC verrà quindi terminata la procedura di selezione dei punti di controllo e comparirà un nuovo pannello contenente i dati ottenuti in output dalla funzione Selezione_punti, nonché gli Input necessari per il tracciamento delle curve.

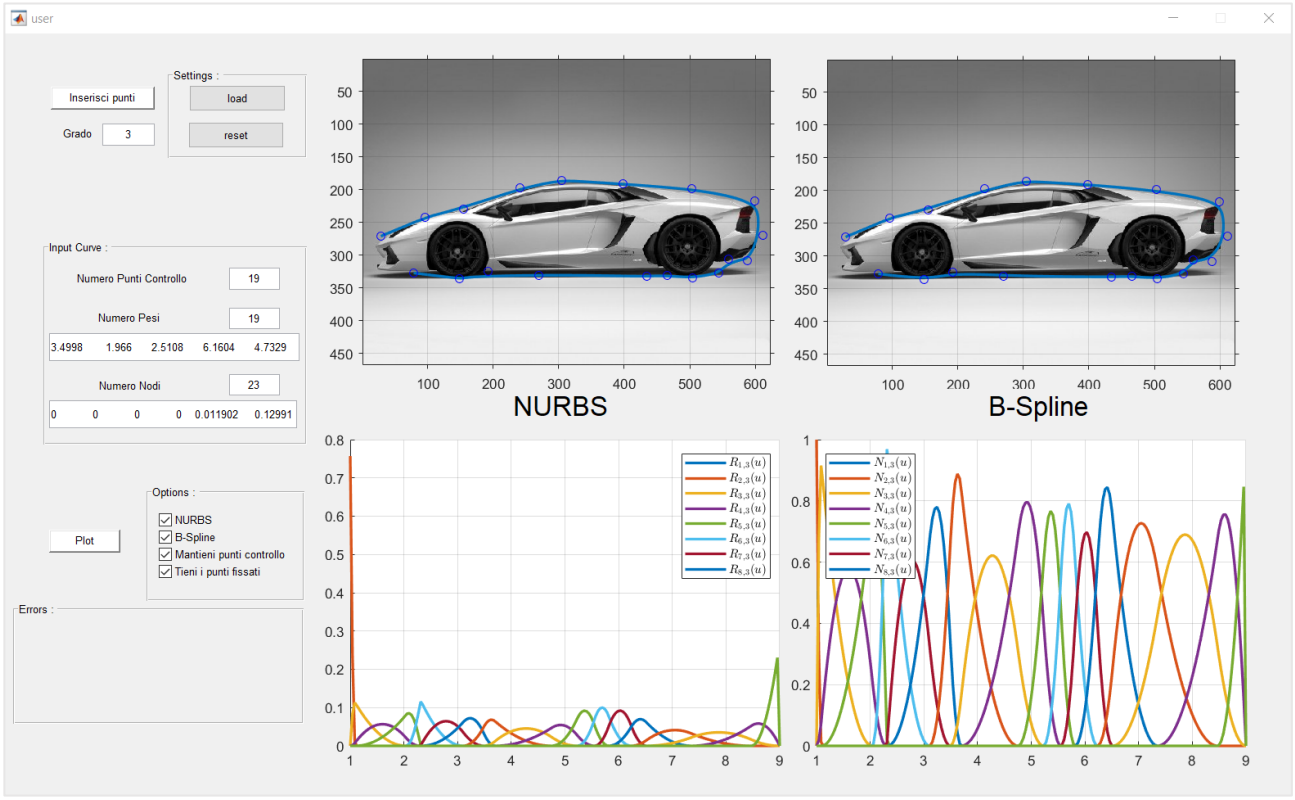


Sono presenti:

- Il campo numero punti di controllo, che restituisce il numero di punti di controllo inseriti dall'utente.
- Il campo numero Pesi che restituisce il numero di pesi calcolati.
- Il campo Pesi contente il valore dei pesi effettivi calcolati dalla funzione.
- Il campo numero Nodi contenente il numero di nodi calcolati.
- Il campo Nodi contente il valore dei nodi effettivamente calcolati dalla funzione.

È possibile per l'utente cambiare questi valori mentre l'interfaccia è a runtime.

Lo step successivo per l'utente sarà quello di cliccare il tasto Plot per tracciare le curve.



Se l'operazione è andata a buon fine, verranno mostrate le curve NURBS sull'asse 1 e 3 e le B-Spline sull'asse 2 e 4, altrimenti verrà restituito un errore nell'error dialog che specificherà il motivo per cui l'operazione non è andata a buon fine e come risolvere.

L'utente potrà modificare i valori dei pesi e dei nodi a patto di non violare i vincoli di integrità dei dati e soprattutto il numero di quest'ultimi.

Per inserire nuovi punti di controllo l'utente dovrà cliccare il tasto di reset, riavviando la procedura dall'inizio.

3.1 Codice Interfaccia

Il codice dell'interfaccia può essere suddiviso in due parti:

1. La prima contente le funzioni di callback necessarie al funzionamento del programma
2. La seconda contenente le callback e gli state di creazione degli oggetti generati da GUIDE.

Analizzeremo solo la prima parte in quanto è stata implementata dal team, la seconda è stata generata automaticamente dal tool GUIDE e contiene funzioni per la disposizione spaziale degli elementi dell'interfaccia, si rimanda il lettore alla documentazione del tool per ulteriori spiegazioni.

3.1.1 *LOADIMAGE_CALLBACK*

Contiene le variabili IMG,img_present la prima per l'immagine e la seconda settata a true quando l'immagine deve essere mostrata. Tramite il comando uigetfile viene aperta la finestra di selezione dell'immagine, imread legge l'immagine e imshow la mostra sul primo e sul secondo asse.

```
function loadimage_Callback(hObject, eventdata, handles)
% hObject    handle to loadimage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global IMG;
global img_present;
[file, path]=uigetfile('*.*');            %apre finestra selezione immagine
nome_img=[path file];
set(handles.errori, 'String', 'Errore nel caricamento immagine');
IMG = imread(nome_img);                                  %legge l'immagine
img_present = true;
set(handles.errori, 'String', ' ');
axes(handles.axes1);                          %importa l'immagine sull'asse 1
hold off;
imshow(IMG);                                             %mostra l'immagine
grid on;
axis on;
hold on;

axes(handles.axes2);                          %importa l'immagine sull'asse 2
hold off;
imshow(IMG);                                             %mostra l'immagine
grid on;
axis on;
hold on;
```

3.1.2 *PUSHBUTTON1_CALLBACK*

Questa callback viene richiamata dal pulsante inserisci punti.

Presenta le variabili P e grado che prendono il loro valore dalla funzione Selezione_Punti() e la variabile clamped settata tramite checkbox.

Oltre che la gestione delle visibilità e degli errori è possibile notare la funzione validateattributes() che effettua un controllo sull'integrità dei valori.

Il numero dei pesi viene calcolato tramite num2str(10*rand(size(P,2),1').

Quello dei nodi viene calcolato con le due funzioni num2str contenute nell' if sulla variabile clamped.

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global P;
global grado;
global clamped;

set(handles.esc_text, 'visible','on');
set(handles.pushbutton1 , 'enable', 'off');
set(handles.loadimage , 'enable', 'off');
set(handles.reset_button , 'enable', 'off');
set(handles.bslpein_butt , 'enable', 'off');

%viene richiamata la funzione per selezionare i punti di controllo
[P,n]=Selezione_Punti(handles);
grado = str2double(get(handles.grado, 'String'));%lettura del grado inserito ↙
dall'utente
clamped = get(handles.clamped, 'Value');

set(handles.esc_text, 'visible','off');
%verifica che il grado della curva sia un numero intero non negativo
set(handles.errori, 'String', 'Il grado deve essere un intero positivo');
validateattributes(grado, {'numeric'}, {'nonnegative','integer','scalar'});
set(handles.errori, 'String', ' ');

set(handles.pushbutton1 , 'enable', 'on');
set(handles.loadimage , 'enable', 'on');
set(handles.reset_button , 'enable', 'on');
set(handles.bslpein_butt , 'enable', 'on');

set(handles.panelvalue , 'visible', 'on');
set(handles.numpuntic_text , 'visible', 'on');
set(handles.numpuntic , 'visible', 'on');
set(handles.numpesi_text , 'visible', 'on');
set(handles.numpesi , 'visible', 'on');
set(handles.numnodi_text, 'visible', 'on');
set(handles.numnodi, 'visible', 'on');
set(handles.pesi , 'visible', 'on');
set(handles.pesi, 'string', num2str(10*rand(size(P, 2), 1)'));
set(handles.nodi, 'visible', 'on');

if clamped
    set(handles.nodi, 'string', num2str([zeros(grado + 1, 1)' sort(rand(size(P, 2) - ↙
grado - 1, 1))' ones(grado + 1, 1)']));
else
    set(handles.nodi, 'string', num2str(sort(rand(size(P, 2) + grado + 1, 1))'));
end
% Numero pesi da inserire
numpesi_real = n;
set(handles.numpesi, 'String', numpesi_real);
numnodi_real = n + grado +1; %calcolo il numero di nodi da inserire
set(handles.numnodi, 'String',numnodi_real);
```

3.1.3 *BSPLINE_BUTT_CALLBACK*

Ha come variabili P,grado;IMG,img_present,pthold,bson,nbson

Bson e nbson tramite get acquisiscono il valore delle checkbox B-Spline e NURBS rispettivamente.

Tramite str2num vengono convertiti i valori dei nodi e dei pesi presenti nei rispettivi campi della GUI.

Resetaxis pulisce tutti gli assi, successivamente se img_preset è settato a true viene mostrata l'immagine e se nbson è settato a true vengono stampate le NURBS tramite DeBoor_NURBS.

È presente un controllo sul valore della checkbox pthold in base al quale viene scelto se stampare o no i punti di controllo.

B_Spline_N calcola le funzioni B-Spline, mentre NURBS_R calcola le NURBS.
Il ciclo for serve per disegnare le funzioni di base sull'asse 3.

Il comando legend crea una legenda con etichette descrittive per ogni dato stampato.

```
function bslpein_butt_Callback(hObject, eventdata, handles)
% hObject    handle to bslpein_butt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global P;
global grado;
global IMG;
global img_present;
global pthold;
global bson;
global nbson;
bson=get(handles.bscheck, 'Value');
nbson=get(handles.nbscheck, 'Value');

t = str2num(get(handles.nodi, 'String'));%vettore dei nodi inserito dall'utente
w = str2num(get(handles.pesi, 'String'));
    resetaxis(handles);
    axes(handles.axes1);
    cla reset;

    if(img_present)
        imshow(IMG);
    end

if(nbson) %checkbox per NURBS
    DeBoor_NURBS(P,grado,t,w,handles);
    pthold=get(handles.pointholder, 'Value');%checkbox stampa anche i punti
    if pthold
        plot(P(1,:),P(2,:),'ob');%stampa solo i punti di controllo
    end
    axes(handles.axes3);
    cla reset;
    grid on;
    axis on;
    hold on;
    N = B_Spline_N(grado, t, size(P, 2));%calcola le funzioni base B-Spline
    R = NURBS_R(N, w);
    for j = 1 : size(P, 2)          %disegna le funzioni di base sull'asse 3
        plot(handles. axes3, linspace(1, 9, 10*size(P, 2)), R(:, j),  'LineWidth',2);
        legendInfo{j} = strcat('$R_{', num2str(j), ',' ,num2str(grado),'}(u)$');
    end
    if(size(legendInfo) < 8)
        legend(legendInfo, 'interpreter', 'latex', 'location', 'best');
    else
         legend(legendInfo(1:8), 'interpreter', 'latex', 'location', 'best');
    end
else
    cla reset;
    grid on;
    axis on;
    hold on;
end
```

La restante parte del codice differisce da quella fino ad ora discussa a discapito dell'utilizzo della funzione DeBoor anziché DeBoor_NURBS e della stampa sul quarto asse.

```
% SECONDA FUNZIONE (B-SPLINE)
axes(handles.axes2);
cla reset;
axes(handles.axes2);
cla reset;

if(img_present)
    imshow(IMG);
end

if(bson)%checkbox per B-Spline
       DeBoor(P,grado,t,handles);
    if pthold
        plot(P(1,:),P(2,:),'ob');
    end

    axes(handles.axes4);
    cla reset;
    grid on;
    axis on;
    hold on;
    N = B_Spline_N(grado, t, size(P, 2));
    for j = 1 : size(P, 2)                %disegna tutte le funzioni di base
        plot(handles. axes4, linspace(1, 9, 10*size(P, 2)), N(:, j),  'LineWidth',2);
        legendInfo{j} = strcat('$N_{', num2str(j), ',' ,num2str(grado),'}(u)$');
    end
    if(size(legendInfo) < 8)
        legend(legendInfo, 'interpreter', 'latex', 'location', 'best');
    else
        legend(legendInfo(1:8), 'interpreter', 'latex', 'location', 'best');
    end
else
    cla reset;
    grid on;
    axis on;
    hold on;
end
```

3.1.4 *BSPLINE_BUTT_CALLBACK*

Ha come variabili img_present.

Quest'ultima viene settata a false in modo da non essere mostrata.

Resetaxis pulisce i quattro assi, successivamente tutti i valori :

(numpuntic ,numpesi, pesi, numnodi, nodi) vengono posti a 0.

```
function reset_button_Callback(hObject, eventdata, handles)
% hObject    handle to reset_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global img_present;
img_present = false;

%reset degli assi: viene gestito da una funzione esterna per garantire il
%riuso.
resetaxis(handles);
%---------------------------------
% setto tutto a 0
set(handles.numpuntic, 'String', '0');
set(handles.numpesi, 'String', '0');
set(handles.pesi, 'String', '0');
set(handles.numnodi, 'String', '0');
set(handles.nodi, 'String', '0');
```

3.1.5 *BSCHECK_CALLBACK & NBSCHECK_CALLBACK*

Entrambe effettuano un controllo sul valore negato di bscheck/nsbcheck e nel caso la casella non sia spuntata (not bscheck/nbscheck) resettano rispettivamente gli assi 2 e 4 nel caso di bscheck e 1 e 3 nel caso di nbscheck.

```
% --- Executes on button press in bscheck.
function bscheck_Callback(hObject, eventdata, handles)
% hObject    handle to bscheck (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if(~(get(handles.bscheck, 'Value')))
    axes(handles.axes2);
    cla reset;
    grid on;
    axis on;
    hold on;
    axis([0 100 0 100]);

    axes(handles.axes4);
    cla reset;
    grid on;
    axis on;
    hold on;
    axis([0 100 0 100]);
end

% Hint: get(hObject,'Value') returns toggle state of bscheck

% --- Executes on button press in nbscheck.
function nbscheck_Callback(hObject, eventdata, handles)
% hObject    handle to nbscheck (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if(~(get(handles.nbscheck, 'Value')))
    axes(handles.axes1);
    cla reset;
    grid on;
    axis on;
    hold on;
    axis([0 100 0 100]);

    axes(handles.axes3);
    cla reset;
    grid on;
    axis on;
    hold on;
    axis([0 100 0 100]);
end
```

3.2 BackEnd

Il programma è interamente implementato in matlab insieme all'ausilio del tool GUIDE per la creazione dell'interfaccia utente.

I file presenti nella cartella dell'elaborato sono rispettivamente:

Nome	Ultima modifica	Tipo	Dimensione
imgtest	16/06/2021 11:37	Cartella di file	
Algoritmo_DeBoor.m	15/06/2021 12:30	MATLAB Code	6 KB
B_Spline_N.m	14/06/2021 17:22	MATLAB Code	4 KB
DeBoor.m	15/06/2021 12:06	MATLAB Code	3 KB
DeBoor_NURBS.m	15/06/2021 12:05	MATLAB Code	4 KB
InserimentoNodi.m	14/06/2021 17:00	MATLAB Code	2 KB
NURBS_R.m	14/06/2021 17:20	MATLAB Code	2 KB
resetaxis.m	15/06/2021 10:32	MATLAB Code	1 KB
Selezione_Punti.m	15/06/2021 12:13	MATLAB Code	4 KB
user.fig	15/06/2021 17:16	MATLAB Figure	106 KB
user.m	16/06/2021 11:11	MATLAB Code	19 KB

- Imgtest, cartella contenente le immagini da caricare nell'interfaccia. L'utente può aggiungerne delle nuove, anche se è consigliabile scegliere file di estensione .jpg in modo tale da non dover gestire la trasparenza del canale alfa (Nota bene: i file con trasparenza verranno lo stesso caricati e sarà possibile lavorarci, ma il livello di trasparenza sarà sostituito da un livello nero). Algoritmo_DeBoor, contenente alcune implementazioni necessarie al funzionamento di DeBoor.m e DeBoor_NURBS.m
- DeBoor.m
- DeBoor_NURBS.m
- InserimentoNodi.m
- NURBS_R.m
- Resetaxis.m
- Selezione_Punti.m
- User.fig modificabile tramite GUIDE
- User.m contenente il codice dell'interfaccia utente.

3.2.1 *SELEZIONE_PUNTI*

Il file contiene la funzione Selezione_Punti che restituisce in output la matrice Punti_C contenente le cordinate dei punti di controllo e il numero num_p di punti di controllo scelti dall'utente. La funzione permette di selezionare i punti di controllo dal grafico, ogni inserimento avviene cliccando sull'asse1 con il tasto sinistro del mouse e l'inserimento termina alla pressione del tasto ESC sulla tastiera.

Dopo aver abilitato gli assi ,vengono inizializzate le variabili necessarie alla selezione dei punti di controllo.

Nel dettaglio viene posto num_p=0 e inizializzati x e y, ovvero i due vettori che andranno a contenere le coordinate dei punti di controllo.

La variabile exit_click inizializzata a false indica la condizione di uscita, ovvero la pressione del tasto ESC da parte dell'utente e vengono settati gli assi 2,3,4 come non clickabili dall'utente durante questa fase. L'utente infatti potrà selezionare i punti soltando dal primo asse.

Segue un ciclo di acquisizione dei punti di controllo tramite la funzione ginput(1) che salva le coordinate dei punti in x_t e y_t ed è inoltre sensibile alla pressione del tasto ESC.

Viene effettuato un controllo sui limiti degli assi, ovvero se l'utente proverà a selezionare punti al di fuori dell'asse 1 verrà stampato a video un messaggio di errori nel error dialog.

Altrimenti verranno stampate sull'asse le coordinate del punto selezionato, salvate nei vettori x e y e verrà incrementato num_p di 1 e successivamente settato tramite handles nel relativo campo della GUI.

Come ultima operazione viene creata Punti_C matrice di zeri che sarà riempita con i vettori x e y.

```
function [Punti_C,num_p] = Selezione_Punti(handles)

%--------------------------------------------------------------------------
%DESCRIZIONE FUNZIONE
%--------------------------------------------------------------------------
%Questa funzione permette di selezionare i punti di controllo dal grafico.
%Ogni inserimento avviene cliccando sul grafico con il tasto sinistro del
%mouse.
%L'inserimento termina premendo il tasto esc sulla tastiera.
%--------------------------------------------------------------------------
%OUTPUT
%--------------------------------------------------------------------------
%La funzione restituisce:
%- la matrice Punti_C contenente le coordinate dei punti di controllo.
%- il numero num_p di punti di controllo.
%--------------------------------------------------------------------------

axis on; %Abilita l'asse
grid on; %Abilita la griglia
hold on; %Mantiene tutte le modifiche successive sugli assi

%Inizializzazione
num_p=0;
x=[];
y=[];
exit_click=false; %condizione di uscita
set(handles.axes2,'HitTest','off');
set(handles.axes3,'HitTest','off');
set(handles.axes4,'HitTest','off');

%Il ciclo permette di acquisire una serie di punti di controllo, il ciclo
%termina quando viene premuto il tasto esc sulla tastiera.
%I punti raccolti saranno inseriti poi all'interno della matrice restituita
%in output.


while exit_click==false
    [x_t,y_t,esc]= ginput(1);                                          %Aziona
il puntatore sul grafico per la selezione dei punti di controllo.
     xlim= get(gca,'XLim');                                            %
Prenede i limiti degli assi in modo da ignorare i punti presi al difuori del
grafico
     ylim= get(gca,'YLim');                                            %
Prenede i clear alllimiti degli assi in modo da ignorare i punti presi al difuori
del grafico
    if esc==27                                                         %il
tasto Esc restituisce il valore 27 nel ginput.
        exit_click=true;
    elseif (x_t<xlim(1) | x_t>xlim(2) | y_t<ylim(1) | y_t>ylim(2))     %
Controlla se il valore acquisito è nei limiti
          set(handles.errori, 'String', 'Puoi inserire punti solo dentro il primo
asse');
    else
        plot(x_t,y_t,'bo');                                            % il
punto di controllo selezionato viene mostrato sul grafico
        x=[x,x_t];                                                     % la
nuova coordinata x viene posta nel vettore delle coordinate x
        y=[y,y_t];                                                     % la
nuova coordinata y viene posta nel vettore delle coordinate y
        num_p=num_p+1;                                                 % viene
incrementato il numero dei punti di controllo inseriti
        set(handles.numpuntic,'String',num_p);                         %
viene aggiornato sull'interfaccia il numero dei punti di controllo inseriti
    end
end

Punti_C=zeros(num_p,2);                                                %
creazione della matrice dei punti di controllo
[Punti_C]=[x;y];                                                       %
inserimento dei vettori x e y nella matrice
set(handles.errori, 'String', ' ');
hold off;                                                              %
disattiva il mantenimento dei dati sul grafico in caso di nuovi inserimenti.

end
```

3.2.2 *RESETAXIS*

Contiene una semplice procedura resetaxis che quando chiamata resetta i valori del primo, del secondo, del terzo e del quarto asse. È stato deciso in fase progettuale di implementare la funzionalità in un function autonoma in quanto il servizio veniva richiesto in diverse fasi di esecuzione del programma.

```
function resetaxis(handles)
axes(handles.axes1);
cla reset;
grid on;
axis on;
hold on;
axis([0 100 0 100]);

axes(handles.axes2);
cla reset;
grid on;
axis on;
hold on;
axis([0 100 0 100]);

axes(handles.axes3);
cla reset;
grid on;
axis on;
hold on;
axis([0 100 0 100]);

axes(handles.axes4);
cla reset;
grid on;
axis on;
hold on;
axis([0 100 0 100]);

end
```

3.2.3 *NURBS_R*

Contiene la funzione NURBS_R che genera i valori delle funzioni di base della NURBS a partire dalle N funzioni di base della B-Spline. Ha come input le funzioni N di base B-Spline e il vettore die pesi w. Restituisce in output un vettore R bidimensionale contenete le n+1 funzioni di base della NURBS.

Ogni elemento contenuto in N viene moltiplicato per il peso w e successivamente viene diviso per la sommatoria di ogni elemento di N moltiplicato il peso w.

```
function R = NURBS_R(N,w)
%-------------------------------------------------------------------------
% DESCRIZIONE
%-------------------------------------------------------------------------
% La funzione NURBS_R genera i valori delle funzioni di base
% della NURBS a partire dalle funzioni di base N della B-Spline.
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% La funzione ha in input:
%   - Le Funzioni N di base B-Spline
%   - il vettore dei pesi w
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% La funzione restituisce in output il vettore R bidimensionale contenente
% le n+1 funzioni di base della NURBS
%-------------------------------------------------------------------------
R = (N.*w)./sum(N.*w);                                                  %↙
Applicazione dell'algoritmo per il calcolo della funzione di base
```

3.2.4 *B_SPLINE_N*

Contiene la funzione B_Spline che calcola i valori delle funzioni di base a partire dal vettore dei nodi.

Ha in input h, ovvero il grado della curva,k vettore dei nodi e num_p numero dei punti di controllo. Restituisce in output N, vettore bidimensionale contenente la size(k) funzioni di base

Viene creato un vettore Z tramite comando linspace di punti in cui viene calcolata la porta. Viene creata una matrice N di dimensioni size(Z) e num_p e una matrice N_h di dimensioni size(Z,2) e size(k,2) utile come supporto al calcolo delle funzioni di base.

Successivamente viene applicata la formula di ricorsione di Cox-De Boor.

Per le funzioni di base con grado pari a 0 viene ripetuto il ciclo per tutti i nodi-1 dato che l'ultimo non avrà un valore successivo con il quale costruire un intervallo . Se il punto del linespace è maggiore del j-esimo nodo e minore del j+1esimo nodo, mette il valore nella j-esmia colonna della matrice di controllo a 1. Altrimenti lo pone a 0.

Per quanto riguarda le funzioni di base di grado successivo, vengono creati i due addendi add1 e add2 che poi andranno sommati tra di loro. Rispettivamente :

- `add1 = (((z-k(j))/(k(j+t-1)-k(j)))*N_h(i,j)).`
- `add2 =(((k(j+t)-z)/(k(j+t)-k(j+1)))* N_h(i,j+1))`

L'ultima riga di codice salva i valori in N.

```
function N = B_Spline_N(h,k,num_p)
%-------------------------------------------------------------------------
% DESCRIZIONE
%-------------------------------------------------------------------------
% La funzione B_Spline calcola i valori delle funzioni di base a partire
% dal vettore dei nodi.
%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% La funzione ha in input:
%       h: grado della curva
%       k: vettore dei nodi
%       num_p: intero, numero dei punti di controllo
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% La funzione restituisce:
%       N: vettore bidimensionale contenente le size(k) funzioni di base
%-------------------------------------------------------------------------
% DATI UTILIZZATI PER IL TESTING
%-------------------------------------------------------------------------
%h=3;
%k=[0,0,0.25,0.32,0.5,0.81,1];
%num_p=3;
%-------------------------------------------------------------------------


Z= linspace( k(1), k(end), 10*num_p);                                   %↙
vettore di punti in cui viene calcolata la curva

% N deve avere dimensione (size(Z),num_p), per ottenere la dimensione del
% linespace, si utilizza la funzione size che restituisce numero di righe e
% colonne di Z, essendo Z un vettore avrà 1 riga e 10*num_p colonne.
% quindi size(Z)= [1 10*num_p], stessa cosa per le dimensioni di k.

N=zeros(size(Z,2),num_p);                                               %↙
Inizzializzazione del vettore bidimensionale che contiene le funzioni di base
N_h=zeros(size(Z,2),size(k,2));                                         %↙
Matrice di supporto al calcolo delle funzioni di base.

%Formula di ricorsione di Cox-De Boor

for i=1:size(Z,2)
    z = Z(i);                                                           % z è↙
uno dei punti in cui viene calcolata la curva.

%   Funzioni di base grado 0
    for j=1:size(k,2)-1                                                 % ripete↙
il ciclo per tutti i numeri di nodi-1 poichè l'ultimo nodo non avrà un valore ↙
successivo su cui costruire un intervallo
        if z>=k(j)&&z<k(j+1)                                            % se il↙
punto del linespace è maggiore del j-esimo nodo e minore del j+1esimo
            N_h(i,j) = 1;                                               % setta↙
il valore ad 1 nella jesima colonna della matrice di supporto
        else
            N_h(i,j) = 0;                                               %↙
altrimenti lo setta a 0
        end
    end


%   Funzioni di base di grado successivo
    for t=2:h                                                           %↙
iterazione per i gradi successivi
        for j=1:size(k,2)-t
            if N_h(i,j)==0
                add1 = 0;
            else
                add1 = (((z-k(j))/(k(j+t-1)-k(j)))*N_h(i,j));
            end
            if N_h(i,j+1)==0
                add2 = 0;
            else
                add2 =(((k(j+t)-z)/(k(j+t)-k(j+1)))* N_h(i,j+1));
            end
            N_h(i,j)=add1+add2;
        end
    end

end

N = N_h(:,1:num_p);
```

La trattazione DeBoor.m, DeBoor_NURBS.m, InserimentoNodi.m, user.m è stata già effettuata nei capitoli precedenti.

Bibliografia

[1] B. Della Vecchia, Dispense di Grafica Computazionale, 2009/10.

[2] Les Piegl ,On NURBS: A Survey, University of South Florida, 1991.

[3] MatLab , https://it.mathworks.com/help/matlab/creating_guis/about-the-simple-guide-gui-example.html, ultimo accesso in data 08/06/2021.

[4] C de Boor (2001), A Practical Guide to Splines, Revised edition, Applied Mathematical Sciences 27, Springer–Verlag, New York.

[5] Luisa D'amore, Modelli Per la Grafica e Data Science, 2021.

[6] Gerald Farin, From Conics to NURBS: Tutorial and Survey, Arizona State University,