UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

TEAPOT SURFACE

Dalle curve alle superfici di Bézier

Anno Accademico 2022/2023

Alfonso Conte M63001378

Daniele Fazzari M63001384

Indice

1 Introduzione **1**
- 1.1 Computer Graphics . . . 2
- 1.2 Modello interpolante e modello approssimante . . . 3

2 Interpolazione e approssimazione **5**
- 2.1 Interpolazione . . . 5
  - 2.1.1 Interpolazione Polinomiale . . . 6
- 2.2 Approssimazione . . . 7
  - 2.2.1 Approssimazione ai minimi quadrati . . . 7
- 2.3 Polyfit e Polyval . . . 8

3 Curve di Bezier **10**

4 Algoritmo di De Casteljau **14**
- 4.1 Curve di Beziér in MATLAB . . . 16

5 Dalle curve di Bezièr alle B-Spline **20**
- 5.1 Il Fenomeno di Runge . . . 20
- 5.2 Interpolazione polinomiale a tratti . . . 23
- 5.3 Curve B-SPLINE . . . 26
- 5.4 Proprietà delle curve B-Spline . . . 28

6 Curve NURBS **32**
  - 6.0.1 Formulazione matematica delle curve NURBS . . . 33
  - 6.0.2 Prorpietà delle curve NURBS . . . 34
  - 6.0.3 Rappresentazione di una circonferenza . . . 36
  - 6.0.4 Rappresentazione teapot con curva NURBS . . . 38
- 6.1 Confronto curve di Bézier, B-Spline e NURBS . . . 39

7 Superfici di Bézier **40**
- 7.1 Cubic Bézier Surface e Teapot . . . 41

Capitolo 1

Introduzione

Con l'affermarsi della simulazione computazionale come strumento d'indagine scientifica, la quantità di dati con cui si ha a che fare è diventata tale da rendere necessaria l'implementazione di metodi matematici (e algoritmi numerici) per la visualizzazione interattiva, efficiente e dettagliata dei dati raccolti (Big Data engineering). Difatti, l'analisi in natura di un fenomeno non è semplicemente determinata dalla capacità di raccogliere "buoni dati" ma anche dalla capacità di saperli interpretare. Inoltre in ambito ingegneristico non è raro che la relazione tra due o più grandezze sia ricavata a partire da un numero finito di valutazioni o misure ottenute generalmente tramite esperimenti, indagini o rilevamenti diretti. Difatti la **Data science** è la scienza che estrae valore dai dati, basata su metodi scientifici e tecniche di analisi degli stessi, al di fine trasformarli in una informazione utile.

Si pone quindi il problema di costruire un modello matematico che descriva e rappresenti in modo attendibile un insieme di dati (inizialmente rappresentabili, ad esempio, tramite un insieme discreto e finito di punti di un piano cartesiano, come mostrato in figura 2.1), in maniera tale da:

- **Dedurre nuove informazioni utili sulla base dei dati già noti** (si pensi ad esempio allo studio del *trend* nel settore dell'economia)
- **Rappresentare dati in modo più conciso** rispetto alla semplice descrizione puntuale degli elementi (si pensi ai modelli di *Clustering*, che condensano insiemi di oggetti in pochi elementi i quali contengono il contenuto statistico rappresentativo del corrispettivo insieme di oggetti al quale appartengono).

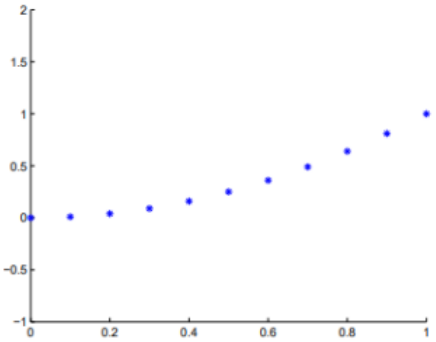


Figura 1.1: **diagramma a dispersione**

Per fare ciò si fa ricorso ad una curva o ad una funzione matematica che possa al meglio descrivere le relazioni presenti tra i dati, ovvero si vuole poter tracciare il grafico di una funzione conoscendone soltanto il valore su un insieme finito di punti della stessa. Tale processo prende il nome di ***Curve fitting***.

1.1 Computer Graphics

La Data Science si espande su un numero elevato di settori (bancario, manifatturiero o sanitario), tra cui quello delle **Computer Graphics** (Computer grafica o grafica computerizzata, in italiano). Per Computer Graphics si intende quella vasta disciplina informatica che ha per oggetto la creazione e la manipolazione di immagini e video digitali, per mezzo di un elaboratore. Il problema principale della computer grafica è sempre stato relativo all'elaborazione dei dati, in quanto c'è la necessità di rappresentare a partire da essi, il profilo che si vuole ottenere attraverso un modello matematico. Tale elaborato si soffermerà proprio su tali aspetti, presentando vantaggi e svantaggi di differenti modelli matematici utilizzabili per la rappresentazione digitale del profilo (anche superficiale) di un oggetto (in particolare sarà utilizzata l'immagine di una teiera, mostrata in figura 1.2) applicando tali tecniche e sfruttando l'ambiente Matlab.

Figura 1.2: **Teapot**

1.2 Modello interpolante e modello approssimante

Ci troviamo dunque a dover scegliere in che modo trasformare i dati acquisiti in una funzione numerica (o curva) da poter analizzare: sono diversi i modelli da utilizzare a seconda del tipo di informazione e dal modo in cui è stata raccolta (ad esempio se abbiamo un numero elevato di dati che potrebbero essere affetti da errori di rilevazione oppure se siamo semplicemente interessati all'andamento globale di un fenomeno piuttosto che alla sua valutazione in un dato punto). Si può in generale fare riferimento a due modelli fondamentali:

- **Modello Interpolante**: Modello secondo il quale si costruisce una curva che passa per i punti assegnati, trascurando di conseguenza l'errore presente nei dati, come mostrato in figura 1.3 e 1.4.

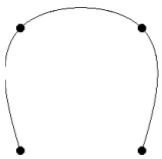


Figura 1.3

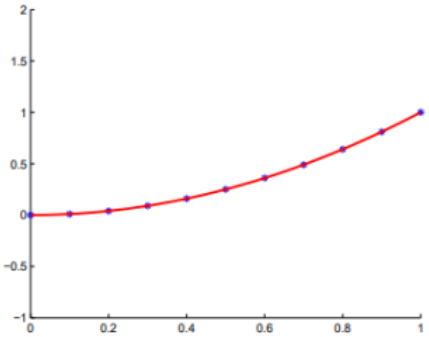


Figura 1.4

- **Modello Approssimante**: Modello che assume non trascurabile l'errore nei dati, dunque non richiede che la funzione passi per i punti assegnati (altrimenti erediterebbe l'errore stesso). Si tratta di un problema a minima distanza, si richiede che la funzione si discosti dai dati di un certo ϵ, ovvero $|f(x_i) - y_i| < \epsilon \quad \forall i = 0...n$. Si mostra un esempio in figura 1.5 e 1.6.

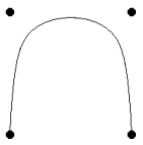


Figura 1.5

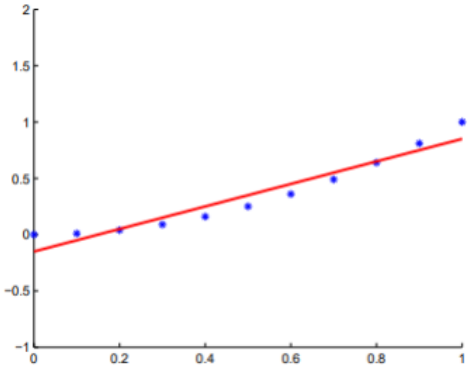


Figura 1.6

Capitolo 2

Interpolazione e approssimazione

2.1 Interpolazione

Il problema generale dell'interpolazione prevede: Dati **n** punti distinti $(x_i, y_i)_{i=1...n}$, costruire una funzione **f(x)** tale che essa nei punti $(x_i)_{i=1...n}$ soddisfi certe condizioni dette ***Condizioni di interpolazione***. Queste ultime sono, in generale, dei vincoli che la funzione f(x) e/o le sue detivate deve soddisfare nei nodi $(x_i)_{i=1...n}$.

Esistono diverse tipologie di interpolazione, che differiscono sulla base delle strategie di elaborazione utilizzate e delle condizioni di interpolazione considerate. Verranno presentati di seguito le condizioni di interpolazione lagrangiane e di Hermite, e la tecnica di interpolazione polinomiale.

Condizioni di interpolazione di Lagrange

Se le condizioni di interpolazione relative alla funzione interpolante coinvolgono esclusivamente i valori che essa assume nei nodi, si parla di condizioni di interpolazione di Lagrange. Dunque dato **F** uno spazio di funzioni di variabile reale o complessa ed assegnati **n** valori distinti (reali o complessi) a $(x_i)_{i=1...n}$ e a $(y_i)_{i=1...n}$, si cerca una funzione f(x) $\in$ F tale che

$$f(x_i) = y_i \quad \forall i = 0...n$$

Condizioni di interpolazione di Hermite

Un passo in avanti rispetto all'interpolazione di Lagrange è compiuto dall'interpolazione di Hermite, la quale ha come vantaggio fondamentale, l'interattività: se la curva ottenuta non è corretta può essere migliorata semplicemente modificando il vettore delle tangenti nei punti di interpolazione. Infatti con essa, dato un insieme di nodi $(x_i)_{i=1...n}$, noti i valori y_i ed eventuali derivate, si ricerca una funzione tale che essa e le sue derivate assumano nei punti x_i gli stessi valori. Formalmente: dati **n** nodi $(x_i)_{i=1...n}$, **n**

interi positivi $(d_i)_{i=1...n}$ (che rappresentano l'ordine di derivazione desiderato per ciascun punto, tali che $\sum_{i=1}^{n} d_i = m$) ed **m** valori $(y_i^j)_{i=0..n\ j=0...d_i-1}$ (che rappresentano il valore della funzione nei punti x_i ed i valori delle rispettive derivate), si vuole determinare una funzione f(x) tale che:

$$f^{(j)}(x_i) = y_i^j \quad \forall i = 0...n, j = 0...d_i - 1$$

Dunque con le condizioni di interpolazione di Hermite si richiede che per ogni nodo sia assegnata almeno una condizione e che, se è assegnata una specifica condizione in un nodo sulla derivata di ordine **q**, allora devono essere necessariamente assegnate in quel nodo tutte le condizioni sulle derivate di ordine inferiore.

2.1.1 Interpolazione Polinomiale

Per tradurre matematicamente la curva che si vuole rappresentare occorre stabilire lo spazio di funzioni in cui ci si muove, mentre fino ad ora abbiamo parlato di una generica funzione interpolante senza specificarne la forma (che in realtà determina il tipo di interpolazione). In questo elaborato tratteremo proprio l'interpolazione polinomiale in cui si richiede che la funzione sia un polinomio di un certo grado. Non a caso, i polinomi sono estremamente vantaggiosi dal punto di vista computazionale e presentano differenti forme rappresentative.

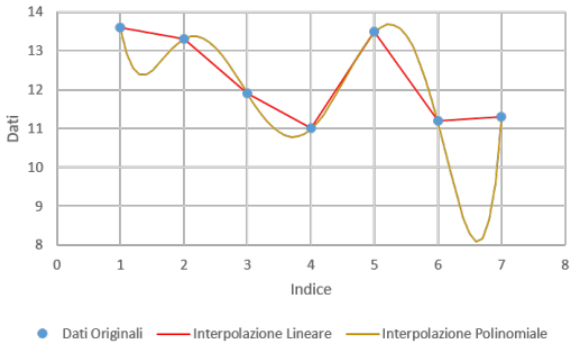


Figura 2.1: Esempio interpolazione polinomiale

Detta quindi f(x) la funzione interpolante, consideriamo n+1 funzioni linearmente indipendenti ϕ_i definite in un certo intervallo [a,b]. Assegnati i valori (x_i, y_i) con $(x_i)_{i=1...n}$ appartenenti all'intervallo di definizione e distinti tra loro, possiamo scrivere f(x) in forma generale come:

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

Il problema si riduce al calcolo dei coefficienti α_i e, ricordando che nel caso di interpolazione polinomiale la generica funzione interpolante f(x) è un certo polinomio p(x), allora le funzioni che costituiscono la

base (ϕ_i) sono anch'esse dei polinomi. Il tipo di rappresentazione delle funzioni (mostrate nella forma generale) conduce a diverse forme di rappresentazione del polinomio:

- **Forma Standard** $p(x) = \sum_{i=0}^{n-1} a_i x^i$
  con α_i coefficienti da determinare.

- **Forma di Lagrange** $p(x) = \sum_{i=1}^{n} y_i l_i(x)$
  con $l_i = \prod_{j=0, j\neq i}^{n} \frac{(x-xj)}{(x_i-xj)}$.

- **Forma di Newton** $p(x) = \sum_{i=0}^{n-1} \alpha_i \phi^i$
  con α_i **differenza divisa** di ordine i-1 (dove per differenza divisa di ordine n-esimo si intende $g(x_0, ..., x_n) = \frac{g(x_0,...,x_{n-1}) - g(x_1,...,x_n)}{x_0 - x_n}$) e $\phi_i = \prod_{j=1}^{i}(x - x_j)$.

In ciascuna rappresentazione le funzioni base sono i polinomi che tengono conto delle condizioni richieste dal modello. Da notare, che non è necessario conoscere tutta la base per la rappresentazione del polinomio ma sono sufficienti i suoi coefficienti.

2.2 Approssimazione

Il problema dell'approssimazione invece prevede, data una successione di n punti distinti $(xi, yi)_{i=1...n}$, la determinazione di una funzione f(x) tale che nei nodi $(xi)_{i=1...n}$ non assuma i valori $(yi)_{i=1...n}$ ma si scosti poco da essi.

2.2.1 Approssimazione ai minimi quadrati

Uno dei possibili metodi di approssimazione è quella ai **minimi quadrati per dati discreti** in cui si sceglie una funzione approssimante f(x) in modo da minimizzare la quantità:

$$\sum_{i=1}^{n} [f(x_i) - y_i]^2 \quad \forall i = 0...n \tag{2.1}$$

Dove tale quantità viene detta **scarto quadratico medio** o **distanza euclidea** (2.1). Quando i dati sono poco accurati e in numero elevato è meglio considerare la minimizzazione dello **scarto quadratico pesato** (2.2) introducendo dei pesi wi:

$$\sum_{i=1}^{n} w_i [f(x_i) - y_i]^2 \quad \forall i = 0...n \tag{2.2}$$

Nella pratica si ha a che fare con funzioni f(x) parametriche, per cui il problema si riduce a determinare i parametri tali che sia minima la distanza dei punti dalla curva. Per evitare di ottenere un fascio di

curve è necessario avere un numero di punti sperimentali che sia maggiore del numero di parametri delle curva stessa, ovvero il problema deve essere **sovradeterminato** (nel caso dell'interpolazione polinomiale l'obiettivo è la ricerca dei coefficienti per minimizzare la distanza, per questa ragione è necessario che il grado p del polinomio cercato sia al massimo pari al numero di punti da interpolare (x_i, y_i) meno 1, ovvero $p \leq n - 1$).

2.3 Polyfit e Polyval

A volte avendo a disposizione dei dati sperimentali o numerici, è utile individuare il polinomio di grado assegnato che meglio li approssima nel senso dei minimi quadrati (minimizzando cioè lo scarto quadratico medio): è il problema del **Data fitting**.
Matlab consente di farlo con la funzione *polyfit*:

```
>> help polyfit
 polyfit Fit polynomial to data.
    P = polyfit(X,Y,N) finds the coefficients of a polynomial P(X) of
    degree N that fits the data Y best in a least-squares sense. P is a
    row vector of length N+1 containing the polynomial coefficients in
    descending powers, P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X + P(N+1).
```

la funzione polyfit(x,y) restituisce i coefficienti del polinomio di grado p che meglio approssima il set di valori (x,y). Una volta ottenuti i coefficienti del polinomio interpolante grazie a polyfit, è possibile graficare la curva descritta da esso, andando a valutare tale polinomio su di un dato intervallo, tramite il comando *polyval*:

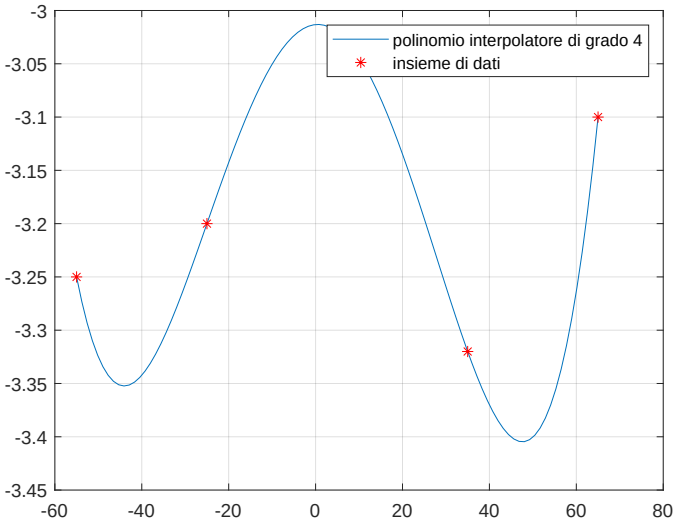
```
>> help polyval
 polyval Evaluate polynomial.
    Y = polyval(P,X) returns the value of a polynomial P evaluated at X. P
    is a vector of length N+1 whose elements are the coefficients of the
    polynomial in descending powers:

        Y = P(1)*X^N + P(2)*X^(N-1) + ... + P(N)*X + P(N+1)
```

Di seguito viene riportato un esempio (*polyfitepolyval.m*) dell'utilizzo delle due funzioni per ricavare l'approssimazione, con relativo output graficato.

```
%Insieme di dati
x=[-55 -25 5 35 65] ;
y=[-3.25 -3.2 -3.02 -3.32 -3.1];
%poli nomio interpolatore di ordine 4
c=polyfit(x,y,4)
%vettore di 100 punti equispaziati nell'intervallo
%tra la prima e l'ultima componente del vettore x
%che serve per generare il plot del polinomio
z=linspace(x(1),x(end),100);
```

```
p=polyval(c,z);
plot(z,p,x,y,'r *');
grid on;
legend('polinomio interpolatore di grado 4','insieme di dati')
```



Capitolo 3

Curve di Bezier

Pierre Bézier, un matematico che lavorava presso la casa automobilistica francese *Renault*, tra gli anni 1950 e 1960 inventò le cosiddette **curve di Bézier**, con lo scopo di ideare una modellazione di linee e superfici matematicamente rigorosa, controllabile e facilmente traducibile in operazioni macchina nel campo della progettazione automobilistica, al fine di automatizzare il processo di design dei veicoli. In altre parole Bézier progettò un modo semplice di creare e levigare le curve dell'auto, per renderla meno spigolosa e adatta alle idee e agli standard che erano stati imposti dalla Renault: l'idea è di definire una curva a partire da un numero finito di punti nello spazio, chiamati **punti di controllo**, in modo che essa segua l'andamento del poligono di controllo individuato da questi punti detto anche **convex hull** (o inviluppo convesso, anche detto in tale applicazione "poligonale di controllo", la quale si ottiene connettendo i punti di controllo con dei segmenti).

Difatti una curva di Bézier è una curva parametrica *C(t)* che è una funzione polinomiale del parametro t. Il grado del polinomio dipende dal numero di punti usati per definire la curva. Il metodo implica l'utilizzo di tali punti di controllo per la produzione di una **curva approssimante**, la quale non passa necessariamente per tali punti ma è "attratta" da essi: ciò rende la curva facilmente modificabile (questo è uno dei principali motivi della popolarità di tale modello matematico).

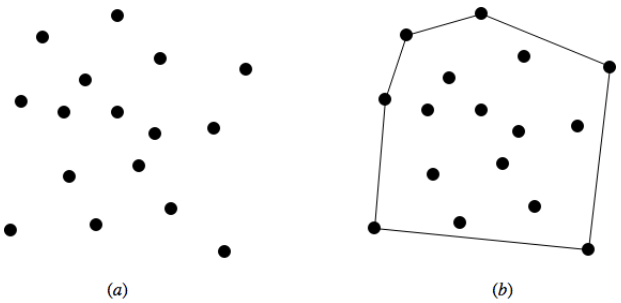


Figura 3.1: set di punti e corrispondente convex hull

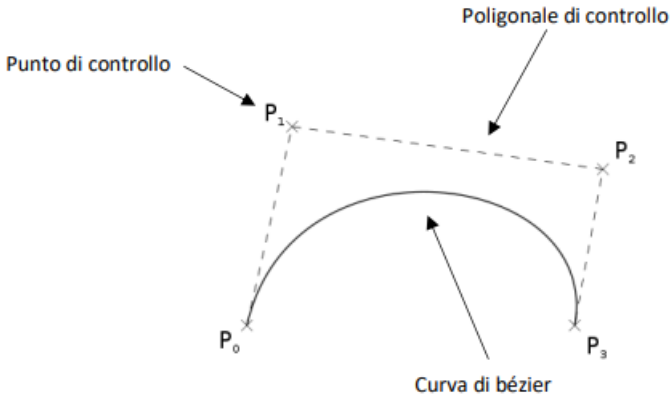


Figura 3.2: Esempio di curva di Bézier con 4 punti di controllo

La curva è completamente contenuta nell'inviluppo convesso dei suoi punti di controllo e quest'ultimi possono essere visualizzati graficamente ed usati per manipolare la curva stessa. Questo è possibile imponendo condizioni sui punti di passaggio e sulla velocità e curvatura della traiettoria che si intende seguire (bisogna introdurre sostanzialmente un controllo sulle derivate nei punti).

Il successo di tale scoperta è tale che ancora oggi le curve di Bézier risultano molto importanti nel campo della grafica computazionale e nella modellizzazione geometrica, difatti sono alla base dei sistemi CAD/CAM. Oltre a ciò le curve di Bézier sono comunemente usate per disegnare loghi e lettere, ampiamente usate nell'animazione, sia come percorsi lungo cui spostare oggetti sia per cambiare le proprietà degli oggetti in funzione del tempo.

In particolare una curva di Bezier è il **centro di massa** (o centro di gravità) del convex hull definito dai punti di controllo. Il centro di massa CM di un oggetto è il punto in cui è concentrato il suo peso, ovvero il punto nel quale si suppone applicata la forza di gravità; nel caso di un insieme finito di punti il CM può essere definito come:

$$CM = \frac{\sum_{i=1}^{n} m_i P_i}{\sum_{i=1}^{n} m_i} \tag{3.1}$$

In particolare nelle curve di Bezier come posizioni P_i vengono utilizzati proprio i punti di controllo.

Ad esempio per 4 punti di controllo l'equazione del centro di massa diventa:

$$CM(t) = \frac{m_0(t)P_0 + m_1(t)P_1 + m_2(t)P_2 + m_3(t)P_3}{m_0(t) + m_1(t) + m_2(t) + m_3(t)} \tag{3.2}$$

Per stabilire chi siano gli m_i occorre analizzare gli attributi delle curve. In particolare dato un insieme finito di punti di controllo $(P_i)_{i=0\ldots n}$, le curve di Bézier sono vincolate a rispettare un certo numero di proprietà, di seguito elencate:

- La curva deve interpolare il punto P_0 (primo punto) e P_n (ultimo punto).

- I punti della curva devono trovarsi interamente all'interno del convex hull.
- La curva è tangente in P_0 al segmento $P_0\ P_1$.
- La curva è tangente in P_n al segmento $P_{n-1}\ P_n$.
- La curvatura in P_0 dipende dai punti P_0, P_1 e P_2.
- La curvatura in P_n dipende dai punti P_{n-2}, P_{n-1} e P_n.
- La derivata k-esima in P_0 dipende dai primi k+1 punti della curva.
- La derivata k-esima in P_n dipende dagli ultimi k+1 punti della curva.

In particolare le funzioni di base m_i hanno le suguenti caratteristiche:

- Assumono un valore non negativo in [0,1]
- Dipendono dal parametro t in [0,1]
- La prima e l'ultima assumono il valore massimo in t=0 e in t=1
- Contribuiscono tutte alla sommma totale, e la loro somma è unitaria.

Nota che il grado del polinomio che defisce la curva di Bézier contenuta nel convex hull è pari ad **n** se sono fissati **n+1** punti di controllo. Date queste premesse, si potrebbe pensare che all'aumentare dei punti di controllo aumenti anche la qualità del modello realizzato: in realtà ciò non è vero a causa del fatto che una interessante proprietà dei polinomi è che le loro **oscillazioni** aumentano all'aumentare del loro grado. Più formalmente possiamo enunciare il teorema di Bernstein: "dato un intervallo [a,b] e fissati in esso n+1 punti con n intero positivo, esiste certamente qualche funzione f(x) continua su tale intervallo con la proprietà che la successione dei polinomi interpolanti $P_1(x), P_2(x), ..., P_n(x)$ (successione di polinomi con grado pari all'indice) non converge uniformemente ad f(x)". Inoltre, se f(x) è sufficientemente regolare, è possibile definire il **resto dell'interpolazione** (o "errore di interpolazione") come $r_n(x) = f(x) - P_n(x)$ e dimostrare che in molti casi l'errore cresce al crescere di n (un esempio di tale casistica sarà fornito successivamente quando si descriverà il fenomeno di Runge).

Risultano dunque evidenti le seguenti problematiche:

- Impossibilità di garantire la convergenza del polinomio alla funzione.
- Legame tra gado del polinomio e numero di punti di controllo.
- Aumento dell'oscillazione al crescere del grado del polinomio: da un lato polinomi di grado basso generano un modello poco attendibile, dall'altro l'utilizzo di polinomi di grado elevato generano un modello poco affidabile.

Per risolvere il problema della convergenza, è possibile definire le funzioni base m_i attraverso i **polinomi di Bernstein** (che fanno si che la curva soddisfi le proprietà sopracitate), costruendo n+1 polinomi, uno per ogni punto di controllo. Ogni polinomio ha grado n e quindi per ogni polinomio ci saranno n+1 incognite (coefficienti). Quindi in totale, per risolvere il sistema lineare per il calcolo dei coefficienti degli n+1 polinomi, ci sarà bisogno di $(n+1)^2$ condizioni (perchè per ognuno degli n+1 punti di controllo ho un polinomio, ed ad ogni polinomio corrispondono n+1 coefficienti incognita). L'equazione della curva di Bézier nella forma di Bernstein è dunque pari a:

$$C(t) = \sum_{i=0}^{n} P_i B_{n,i}(t) \qquad t \in [0,1] \tag{3.3}$$

con $P_0 ... P_n$ punti di controllo e $B_{n,i}(t)$ polinomi di Bernstein di grado n:

$$B_{n,i} = \binom{n}{i} t^i (1-t)^{n-i} \qquad \binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{3.4}$$

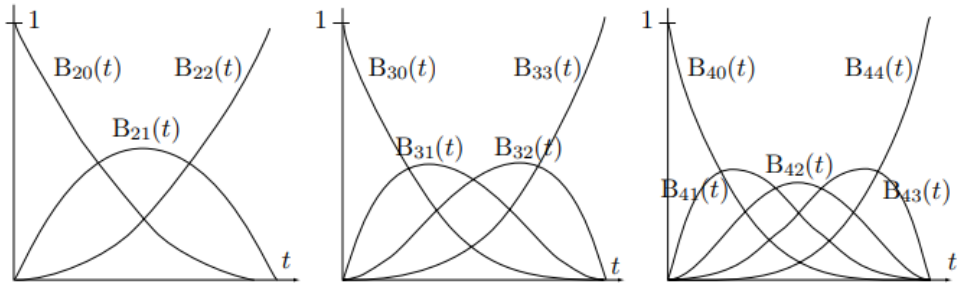


Figura 3.3: Polinomi di Bernstein per n = 2,3 e 4

Vediamo perchè i $B_{n,i}(t)$ devono essere dipendenti dal parametro t. Esaminando prima il polinomio $B_0(t)$, ovvero il peso associato al primo punto di controllo P_0, vogliamo che esso influenzi la curva maggiormente all'inizio (quindi quando t=0) e al crescere di t verso 1 (quindi all'allontanarsi da P_0) $B_0(t)$ dovrebbe scendere a 0 (infatti quando $B_0(t) = 0$ allora il primo punto non influenza più la forma della curva). Quindi la stessa cosa vale per i punti di controllo i-esimi P_i successivi: il peso $B_i(t)$ corrispondente dovrebbe iniziare ad un valore basso (perchè non deve influenzare molto gli altri punti di controllo) per poi raggiungere il suo massimo in corrispondenza del punto di controllo P_i (dove la influenza del peso deve essere massima) per poi scendere nuovamente verso 0. Dunque tali funzioni peso fungono da "funzioni mescolatrici" che mescolano i contributi corrispondenti ai diversi punti di controllo.

Capitolo 4

Algoritmo di De Casteljau

L'**algoritmo di De Casteljau** permette di costruire la curva di Bézier associata al vettore dei punti di controllo assegnato, lavorando su **combinazioni lineari**. Fissato un valore t∈[0,1], esso permette di calcolare il punto corrispondente sulla curva C(t) mediante interpolazioni lineari ripetute a partire dai punti di controllo.

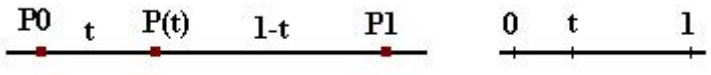
La procedura operativa prevede che ogni lato del poligono di controllo venga diviso in due parti formando un nuovo punto. I nuovi punti calcolati, collegati tra loro, determinano un poligono con n-1 lati. Ognuno di questi n-1 lati viene suddiviso in due parti dando luogo ad una batteria di nuovi punti. Questi nuovi punti, collegati tra loro, determinano un poligono con n-2 lati. Così procedendo si ottiene al termine un solo lato che, suddiviso in 2 parti permette di determinare il punto sulla curva.

In generale, dati n+1 punti, la curva di grado n è data dalla seguente equazione di ricorrenza:

$$P_i^k(t) = (1-t)P_i^{k-1}(t) + tP_{i+1}^{k-1}(t) \qquad t \in [0,1], k = 1...n, i = 0...n-k \tag{4.1}$$

Curva di grado 1

Dati due punti del piano P_0 e P_1

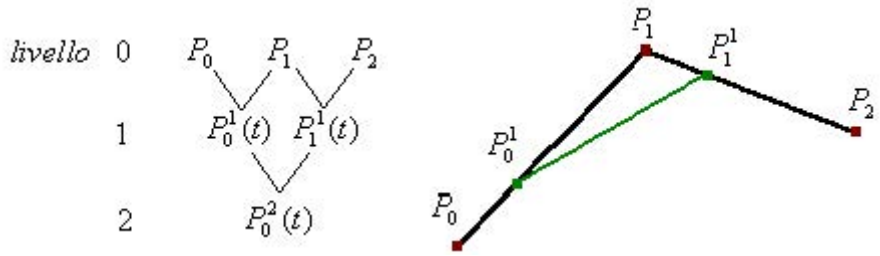


Un punto qualsiasi della curva C(t) compreso tra P_0 e P_1 sarà:

$$C(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1 \tag{4.2}$$

Curva di grado 2

Dati tre punti P_0, P_1 ,P_2 vengono create due curve di supporto di grado 1, date dalle due coppie di punti P_0,P_1 e P_1,P_2. Considerando un valore t∈[0,1], definiamo due punti ulteriori sui due segmenti la cui unione genera un ulteriore segmento denominato poligonale di primo livello. A partire da tale segmento viene individuato un ulteriore punto funzione di t corrispondente alla curva di Bézier di grado 2.



(a) (b)

L'equazione dei punti della curva così ottenuta per sostituzione è:

$$P_0^1(t) = (1-t)P_0^0(t) + tP_1^0(t) \tag{4.3}$$

$$P_1^1(t) = (1-t)P_1^0(t) + tP_2^0(t) \tag{4.4}$$

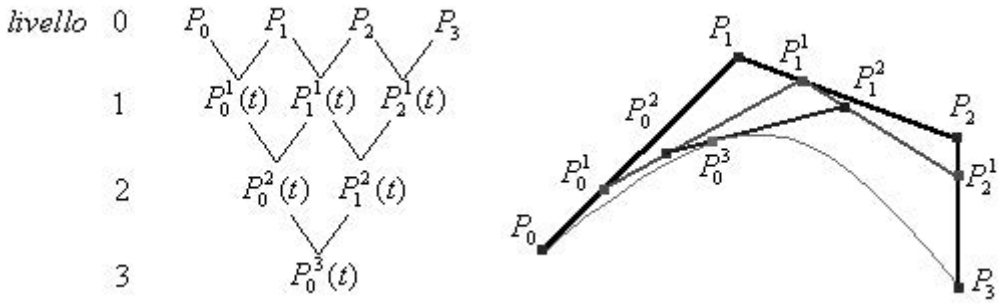
$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t) \tag{4.5}$$

Sostituendo i valori di $P_0^1(t)$ e $P_1^1(t)$ otteniamo l'equazione della curva di Bézier

$$C(t) = P_0^2(t) = (1-t)^2P_0^0 + 2(1-t)tP_1^0 + t^2P_2^0 \tag{4.6}$$

Curva di grado 3

Infine nel caso delle curve di grado 3 il discorso precedente si estende introducento un altro punto a formare il poligono di controllo ottenendo così, infine, una curva definita dalla equazione 4.7.



(a) (b)

$$C(t) = P_0^3 = (1-t)^3 P_0^0 + 3(1-t)^2 t P_1^0 + 3(1-t)t^2 P_2^0 + t^3 P_3^0 \tag{4.7}$$

L'algoritmo presenta le seguenti caratteristiche:

- **Stabilità**: i coefficienti 1-t e t sono compresi tra 0 e 1 e quindi non c'è amplificazione dell'errore
- **Ricorsione**: Data l'applicazione della stessa formula ad ogni passaggio
- **Lentezza**: dovuta alla complessità dell'algoritmo che è dell'ordine di $O(n^2)$ per ogni punto della curva, dato che vi sono due cicli innestati sul vettore dei punti di controllo che viene passato in ingresso.

4.1 Curve di Beziér in MATLAB

Matlab mette a disposizione delle apposite funzioni per la costruzione delle curva di Bézier generando opportunamente i polinomi di Bernsteìn tramite il comando:

```
B = bernsteinMatrix(k,t)
```

Tale funzione restituisce un vettore n-dimensionale, contenente gli n polinomi di Bernstein di grad k=n-1 (dove ricordiamo n essere il numero di punti di controllo)

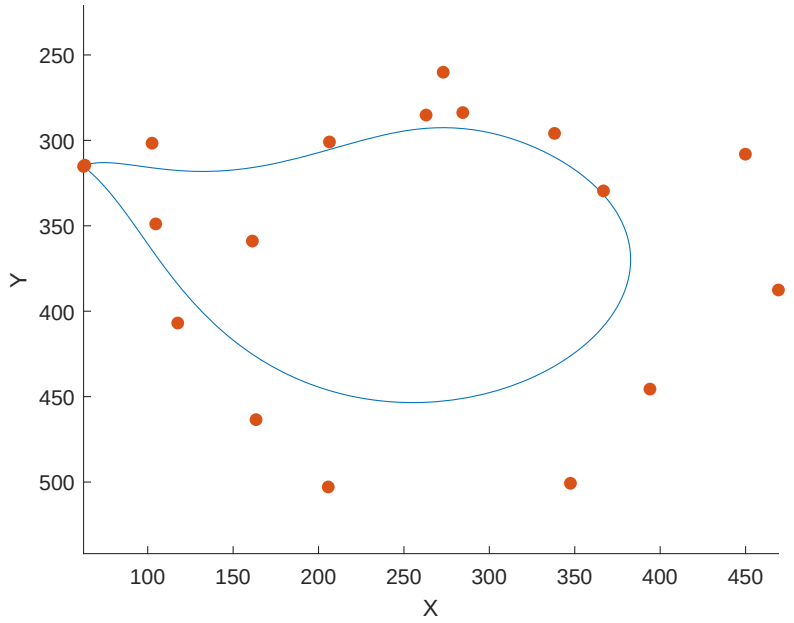
Curve di Beziér 2D

Riportiamo di seguito il codice Matlab per costruire e graficare una curva di Beziér. Il caso che considereremo è la costruzione di una curva di Beziér che approssimi il profilo bidimensionale di una *Teapot*.

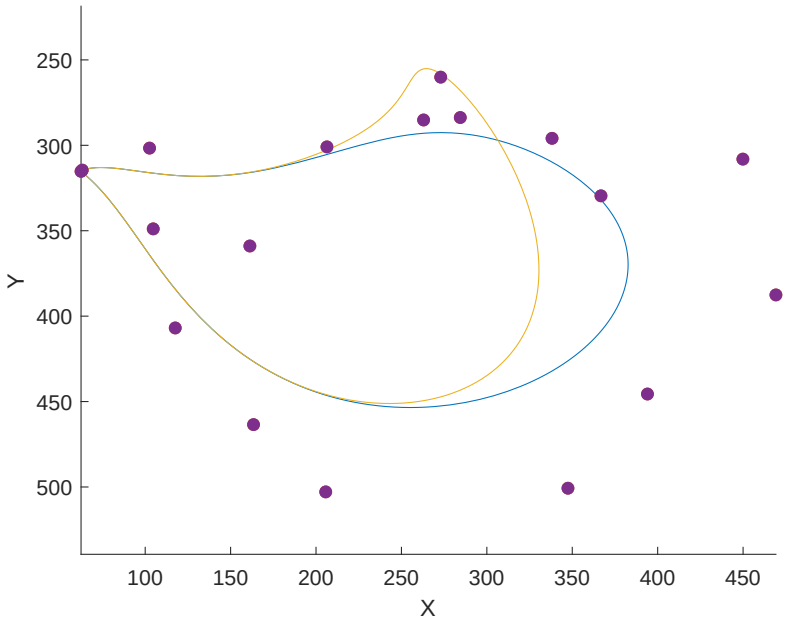
In particolare per ricavare i punti di controllo è stata utilizzata la funzione MAtlab *getpts()*, la quale consente di ricavarli direttamente tramite puntatore del mouse.

```
%Bezier2D.m
load x;
load y;
P=[x y];
syms t
B=bernsteinMatrix(length(P)-1,t);
bezierCurve = simplify(B*P);

figure(1);
hold on;
axis('equal');
set(gca, 'YDir','reverse');
fplot(bezierCurve(1), bezierCurve(2), [0, 1])
xlabel('X')
ylabel('Y')
hold on
scatter(x, y,'filled')
```



Proviamo adesso a cambiare uno solo dei punti di controllo contenuti nel vettore P (in partiolare accediamo al nono punto del vettore). Graficando la nuova curva di Beziér si vede chiaramente come la modifica locale anche di un sol punto abbia un impatto globale sul suo andamento, questo perchè le funzioni di base che sono state impiegate sono definite e non nulle in tutto l'intervallo di definizione della curva, dunque continuano a fornire contributo alla sua rappresentazione. Per superare questo problema è possibile impiegare un modello interpolante o approssimante a tratti.



Curve di Beziér 3D

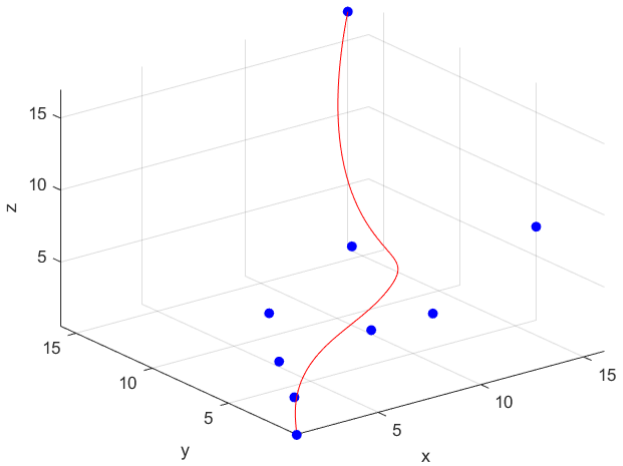
Negli esempi finora riportati abbiamo usato sempre punti di $\mathbb{R}^2$, tuttavia le curve di Bézier sono definite anche per vettori di $\mathbb{R}^3$ e l'algoritmo illustrato precedentemente funziona anche in questo caso. Se ne illustra dunque un caso di applicazione.

```
%Bezier3D.m

%Costruisco la Matrice P dei punti di controllo
P=[1 0.5 0.5;2 2 2;3 5 6;8 1 6; 5 1 6; 10 9 6; 16 5 7; 2 3 4; 15 16 17];

syms t
%Determiniamo la matrice di Bernstein
B=bernsteinMatrix(length(P)-1,t)
Curva=simplify(B*P)

fplot3(Curva(1),Curva(2),Curva(3),[0 1],'r')
xlabel('x')
ylabel('y')
zlabel('z')
hold on
scatter3(P(:,1),P(:,2),P(:,3),'filled','b')
grid on;
hold off;
```



Capitolo 5

Dalle curve di Bezièr alle B-Spline

5.1 Il Fenomeno di Runge

Sia nel caso dell'interpolazione polinomiale, che quello di approssimazione con le curve di Beziér le funzioni di base interpolanti che abbiamo impiegato sono i polinomi. Questi, pur essendo molto vantaggiosi dal punto di vista computazionale sono limitati nel loro utilizzo a causa delle oscillazioni che si verificano all'aumentare del grado, per cui il tentativo di aumentare il numero di nodi potrebbe generare l'effetto contrario rispetto al miglioramento dell'**affidabilità del modello**. Tale considerazione rappresenta una forte limitazione per l'impiego dei polinomi nei processi di interpolazione, in particolar modo per le curve di Beziér dove, come abbiamo visto nei paragrafi precedenti, è richiesto che il grado sia pari al numero di punti di controllo meno uno. Il fenomeno descritto è noto come ***Fenomeno di Runge*** scoperto dal matematico *Cart Runge* mentre studiava il comportamento degli errori di interpolazione polinomiale per approssimare alcune funzioni.

Consideriamo ad esempio la funzione:

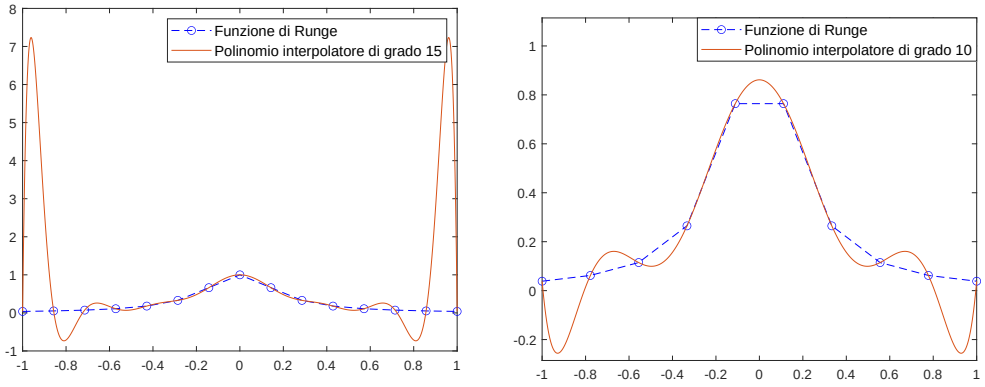
$$f(x) = \frac{1}{1+25x^2} \qquad con \quad x \in [-1,1] \tag{5.1}$$

Proviamo a determinare l'interpolazione polinomiale su un insieme di n punti equistanziati nell'intervallo [-1,1] con un polinomio $P_m(x)$ di grado m$\leq$ n.

```
%Runge.m
n=10
x=linspace(-1,1,n);
runge=@(x) 1./(1+25*x.^2);
y=runge(x);
c=polyfit(x,y,n-1);

%Rappresentiamo graficamente il polinomio e la funzione
```

```
z=linspace(-1,1,1000);
p=polyval(c,z);
plot(x,y,'o--b',z,p);
legend('Funzione di Runge','Polinomio interpolatore di grado n','FontSize',11)
```



(a) *Runge n=15* (b) *Runge n=10*

Figura 5.1

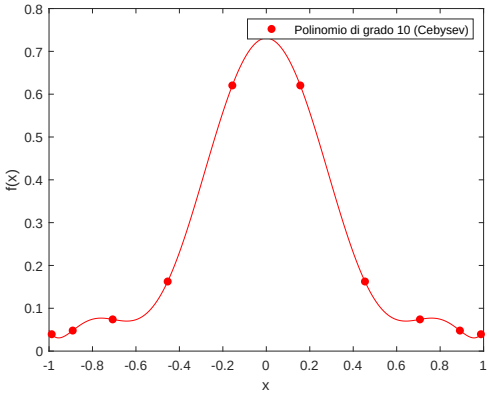
Dalle figure 5.1 si vede chiaramernte l'oscillazione in corrispondenza degli estremi dell'intervallo (da notare la crescita dell'ampiezza di tale oscillazione nel passaggio da n=10 a n=15)

E' possibile considerare uno schema di interpolazione alternativo determinando, in sostituzione a nodi equispaziati nell'intervallo considerato, i cosiddetti **nodi di Gauss-Chebyshev-Lobatto**. Essi altro non sono che le radici dei polinomi di Chebyshev. Infatti per ogni n intero naturale il polinomio n-esimo possiede n radici semplici interne all'intervallo [-1,1]. Una tale n-pla costituisce una buona scelta per una interpolazione degli n punti dell'intervallo in quanto consente una **maggiorazione** dell'errore di interpolazione. Tali nodi sono così definiti:

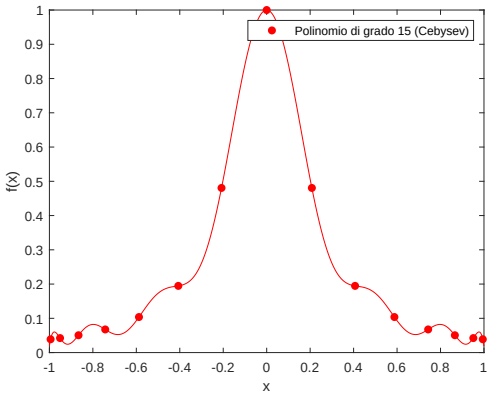
$$x_i := cos(\frac{2i-1}{2n}\pi) \qquad 1 \leq i \leq n \tag{5.2}$$

```
%Cebysev.m
n=15;
runge=@(x) 1./(1+25*x.^2);
x=linspace(-1,1,1e4);
y=runge(x);
ceby= @(i,n) cos(((2*i-1)*pi)./(2*n));
xc=ceby(1:n,n);
yc=runge(xc);
p = polyfit(xc,yc,n-1);
t = linspace(min(x),max(x),1e4);
v = polyval(p,t);
figure(1)
plot(t,v,'r');
```

```
hold on;
scatter(xc,yc,'r','filled');
xlabel('x')
ylabel('f(x)')
legend('','Polinomio di grado 15 (Cebysev)');
hold off;
```



(a) *Cebysev n=10*



(b) *Cebysev n=15*

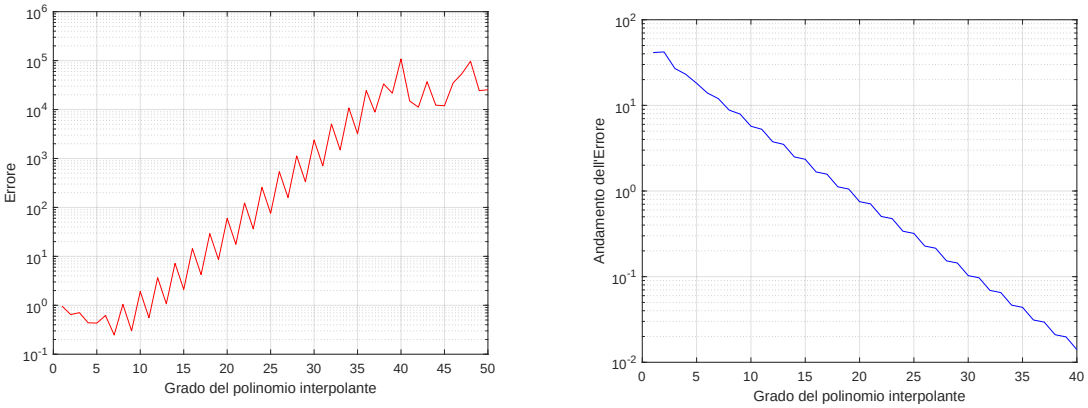
Definiamo a questo punto l'errore tra f(x) e P(x) come $\lim_{n\to\infty, x\in[-1,1]}(max|f(x)-P_n(x)|)$ e valutiamo l'andamento di tale errore (al variare del grado del polinomio interpolante) nei due casi che sono stati presentati.

```
%CASO 1 ErroreRunge.m
err= zeros(50,1);
runge=@(x) 1./(1+25*x.^2);

for i=1:50
    x=linspace(-1,1,i+1);
    y= runge(x);
    p = polyfit(x,y,i);
    t = linspace(min(x),max(x),1e4);
    v = polyval(p,t);
    yt= runge(t);
    err(i) =norm(yt-v,inf);
end
figure(1)
semilogy(err,'r');
xlabel('Grado del polinomio interpolante')
ylabel('Errore')
grid on

%CASO 2 Errore_Ceby
err = zeros(40,1);
ceby= @(i,n) cos(((2*i-1)*pi)./(2*n));
```

```
runge=@(x) 1./(1+25*x.^2);
for i = 1:40
    xc = ceby(1:i+1,i+1);
    yc = runge(xc);
    p = polyfit(xc,yc,i);
    t = linspace(min(xc),max(xc),1e4);
    v = polyval(p,t);
    y = runge(t);
    err(i) = norm(y-v);
end
figure(1)
semilogy(err,'b');
xlabel('Grado del polinomio interpolante')
ylabel('Andamento dell''Errore')
```



(a) *Andamento Errore Nodi Equispaziati* (b) *Andamento Errore nodi di Cebysev*

Figura 5.3

Dalle figure 5.3 si vede chiaramente come nel primo caso l'errore cresce al crescere del grado del polinomio interpolante, viceversa risulta evidente che sussiste invece la convergenza puntuale al crescere di n qualora si utilizzino i nodi di Chebyshev.

5.2 Interpolazione polinomiale a tratti

Le tecniche di interpolazione viste precedentemente presentano una serie di problemi nelle applicazioni delle curve di Beziér, quali:

- Oscillazioni della curva nell'intorno degli estremi del dominio.
- Impossibilità di un controllo locale, per cui la modifica di un punto ha forte impatto sull'intera curva.

Ci si chiede, dunque, qualora si possegga un gran numero di punti, anche equispaziati, se sia possibile calcolare un'approssimante di tipo polinomiale per cui al crescere di n si abbia una **convergenza uniforme** di $P_n(x) \to f(x)$.

E' per questa ragione che è richiesta una **base a supporto locale**, ovvero l'introduzione di un modello approssimante/interpolante a tratti. Tale modello consiste nel partizionare l'insieme su cui sono definiti i dati da rappresentare in m sottoinsiemi, e su ognuno di essi si costruisce la relativa funzione interpolante che, nel caso di un polinomio, porta ad una **interpolazione polinomiale a tratti**. Tipicamente si lavora sui sottointervalli con polinomi di grado relativamente basso proprio per evitare oscillazioni. Inoltre, data la definizione di ciascun polinomio solo nel rispettivo sottointervallo, il suo contributo sarà riservato esclusivamente ad esso.

Siano assegnate m+1 osservazioni y_i con i=0...m nei punti (nodi) $a = x_0 < x_1 < ... < x_m = b$, un polinomio interpolante a tratti consiste di m polinomi di grado n«m tali che $p_k(x)$ definito su $[x_k, x_{k+1}]$ soddisfa:

$$p_k(x_k) = y_k \qquad p_k(x_{k+1}) = y_{k+1} \qquad k = 0...m-1 \tag{5.3}$$

Il primo caso è quello delle interpolanti polinomiali a tratti di grado 1 (interpolazione lineare a tratti)

Per fare ciò utilizzeremo la funzione MATLAB interp1 per valutare un banale esempio:

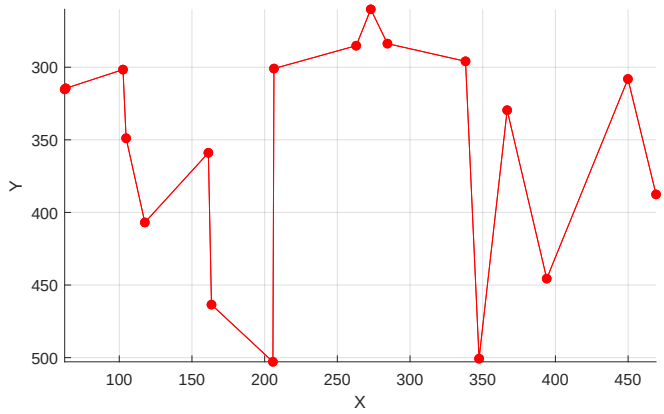
```
>> help interp1
 interp1 1-D interpolation (table lookup)

    Vq = interp1(X,V,Xq) interpolates to find Vq, the values of the
    underlying function V=F(X) at the query points Xq.
```

Riprendiamo il caso di studio relativo al problema di determinazione delle curva associata alla Teapot e applichiamo l'interpolazione polinomiale a tratti (*Interpolazione_polinomiale_a_tratti.m*).

```
load x;
load y;
P=[x y];
x1=sort(x)
t=linspace(x1(1),x1(18),1e4);
f=interp1(P(:,1),P(:,2),t);
figure(1);
hold on;
axis('equal');
set(gca, 'YDir','reverse');
plot(t,f,'r')
hold on
scatter(P(:,1),P(:,2),'or','filled')
xlabel('X')
ylabel('Y')
```

```
grid on
axis tight
```



5.3 Curve B-SPLINE

Le B-splines nascono come evoluzione diretta delle curve di Bezier allo scopo di risolvere il problema del peso "globale" riguardo i punti di controllo. Due sono le esigenze richieste:

- Polinomi di grado moderatamente basso
- Regolarità della funzione risultante (curva "*smooth*")

Ciò è realizzato grazie alle **funzioni Spline**, e le B-splines altro non sono che splines linearmente indipendenti. Si tratta di una tipologia di curva definita "composita", ovvero costruita congiungendo con continuità segmenti adiacenti di curve plinomiali. La tecnica utilizzata è la saldatura di tipo C-2: l'ultimo punto della prima curva coincide col primo punto della seconda curva, e in entrambi i punti si avrà la stessa tangente e curvatura (rispettivamente derivata prima e seconda). Le Curve B-SPLINE sono matematicamente così definite:

$$C(t) = \sum_{i=0}^{n} P_i B_{i,h}(t) \tag{5.4}$$

ove P_i sono gli n+1 punti di controllo, mentre $B_{i,h}(t)$ è l' i-esima funzione B-Spline di grado h definita in $t \in [t_i, t_{i+h+1}]$, intervallo in cui la funzione ha supporto compatto (ovvero il sottoinsieme dei punti del dominio in cui la funzione assume valori non nulli). Di seguito si riportano invece i parametri di una curva B-SPLINE:

- n+1 punti di controllo $P_0 ... P_n$
- Grado h
- Vettore dei nodi $T = (t_0, t_1, ..., t_{n+h+1}) \quad t_i < t_{i+1}$

Con le funzioni B-spline i nodi definiti svolgono un ruolo significativo. In particolare il numero di nodi risulta pari a: n+1+h+1 (ovvero la somma tra il numero di punti di controllo, il grado della curva ed una unità).

In particolare i polinomi $B_{i,h}(t)$ sono dei polinomi sostitutivi rispetto a quelli di Bernstein, calcolati con le formule ricorsive di **De Boor**:

$$B_{i,1}(t) = \begin{cases} 1 & se \quad t_i \leq t < t_{i+1} \\ 0 & altrimenti \end{cases} \tag{5.5}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(t) \tag{5.6}$$

Possiamo dire in generale che le curve B-Spline costituiscono un modello più affidabile rispetto alle curve di Beziér, e ciò è dovuto al fatto che nel caso delle B-Spline il grado dei polinomi interpolanti può

essere scelto a priori fissandolo opportunamente, evitando dunque che l'errore di interpolazione cresca al crescere del grado dei polinomi stessi.

Per plottare una curva B-Spline e le sue funzioni di base si utilizza la seguente funzione Matlab:

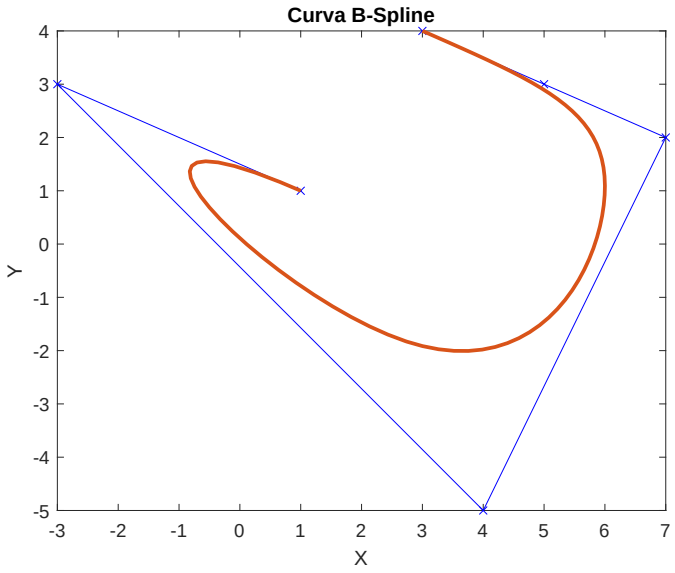
```
bspline(t)   % con t che rappresenta la sequenza di nodi assegnati
```

Riportiamo di seguito il grafico della curva relativo al vettore t = [0 1.5 2.3 4 5];

Inoltre Matlab mette a disposizione un'altra funzione per la rappresentazione di una curva B-Spline cubica a partire da fissati punti di controllo, un esempio è di seguito riportato (*BSpline_ cp.m*)

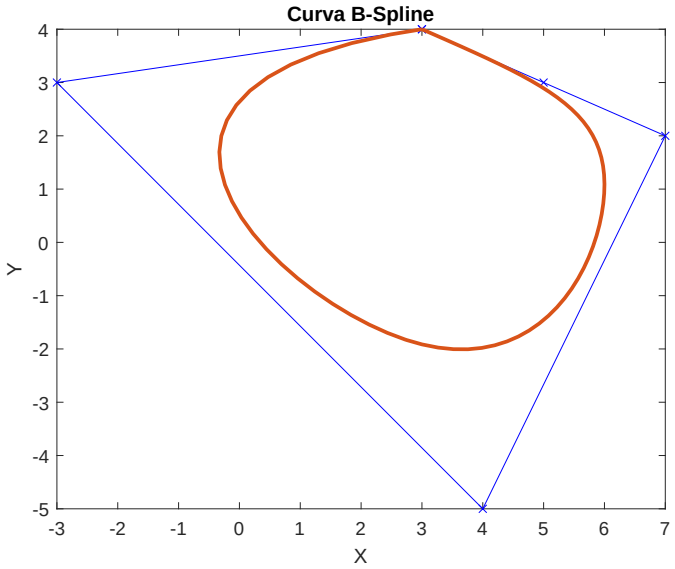
```
%Seleziono i punti di controllo
controlPoints = [3 5 7 4 -3 1; 4 3 2 -5 3 1];
%Scelgo l'intervallo temporale selezioando istante iniziale e finale
tInterval = [0 5];
%Campioni temporali per la traiettoria, specificati come vettore
tSamples = 0:0.01:5;

[q,~,~,pp]=bsplinepolytraj(controlPoints,tInterval,tSamples)
plot(controlPoints(1,:),controlPoints(2,:),'xb-');
title('Curva B-Spline')
xlabel('X')
ylabel('Y')
hold on
fnplt(pp)
```



5.4 Proprietà delle curve B-Spline

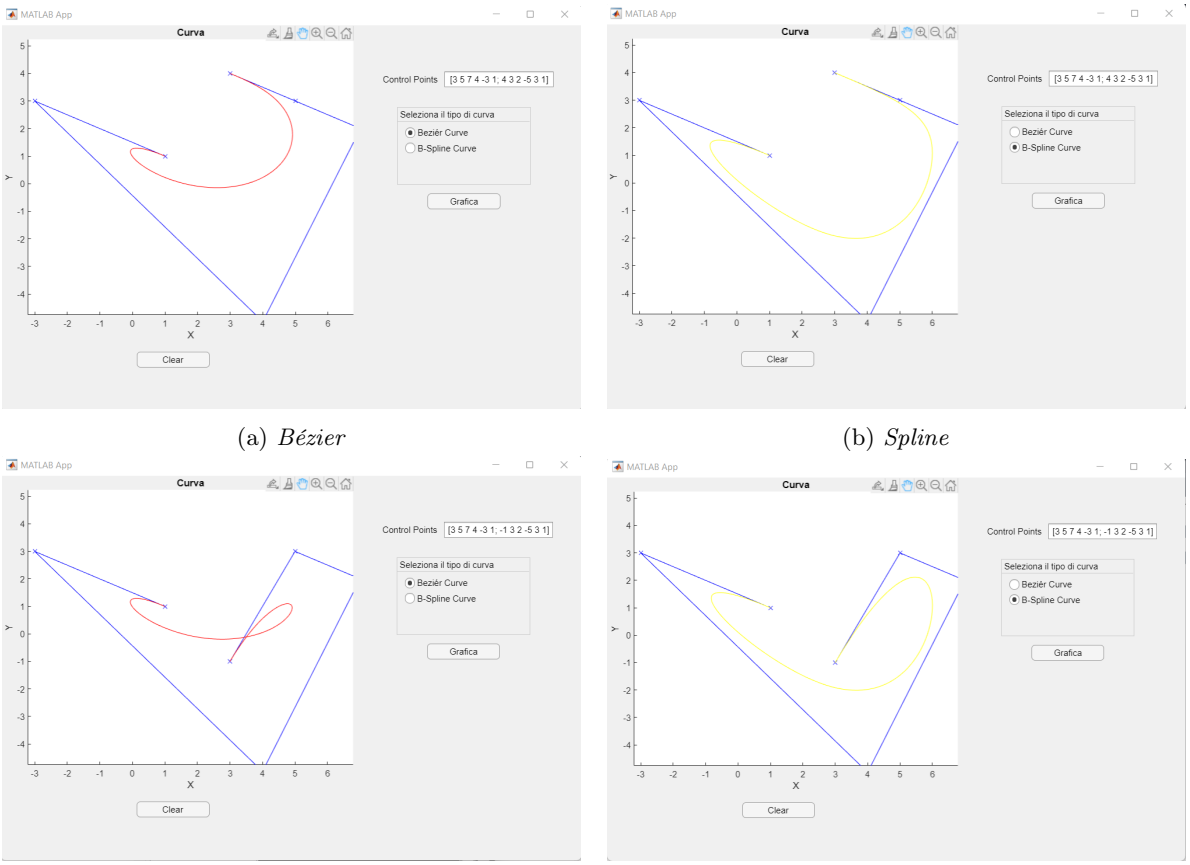
- **Cruva chiusa**: In generale una curva B-Spline non è chiusa, ma può essere resa tale se l'ultimo punto di controllo P_n coincide con il primo P_0.



- **Invarianza**: possiamo traslare, ruotare e deformare la curva applicando trasformazioni geometriche o affini ai punti di controllo, per poi rivalutare la curva evitando che tale operazione venga eseguita per ogni singolo valore delle curva stessa.

- **Controllo locale**: diretta conseguenza del supporto compatto delle B-Spline tale per cui, se P_j cambia, la curva risulta modificata solo in corrispondenza dei punti di definizione della B-Spline $B_{j,h}$.

Consideriamo ad esempio i seguenti punti di controllo: (3,4),(5,3),(7,2),(4,-5),(-3,3),(1,1) e rappresentiamo le corrispondenti Curve B-Spline e Beziér. Procedendo a modificare il punto di controllo (3,4) in (3,-1) si vede chiaramente dalla figura 7.1 come la curva di Beziér sia fortemente affetta da tale cambiamento, mentre la Spline continui a mantenere il suo andamento. Per graficare le due tipologie di curve è stata realizzata una piccola applicazione in Matlab (file BEZIERAPP.mlapp) la quale richiede di inserire il vettore dei punti di controllo e sulla base Bottone relativo al botton group selezionato riporta il grafico della curva di Bézier o della Spline corrispondente.



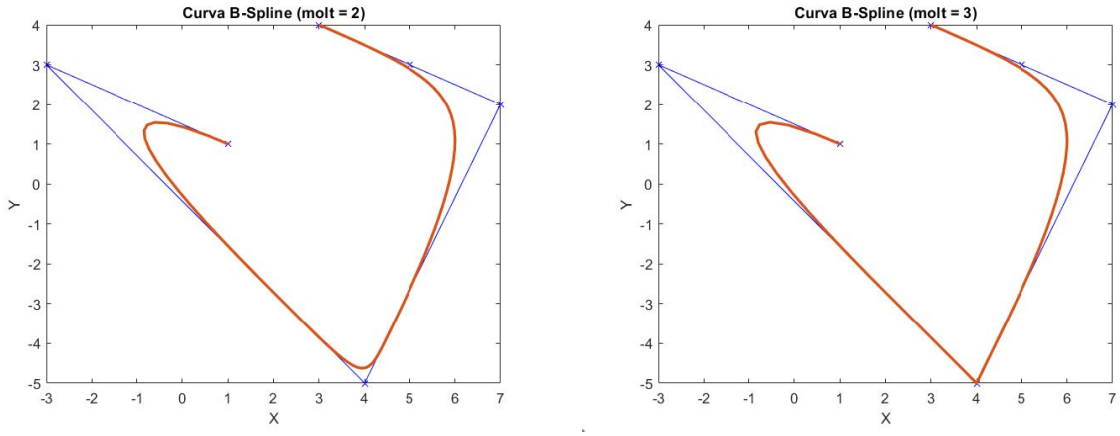
(a) *Bézier* (b) *Spline*

(c) *Bézier* (d) *Spline*

Figura 5.4: Esempio Cambio di un punto di controllo

- E' inoltre possibile modificare la curva in modo che sia tangente al primo e all'ultimo segmento nel primo e nell'ultimo punto di controllo rispettivamente. Per fare ciò il primo e l'ultimo nodo devono avere molteplicità h+1.

- In generale una B-Spline non interpola alcun punto di controllo, ma fissato il grado h, all'aumentare della molteplicità del nodo la curva si avvicina ad esso fino ad interpolare il punto di controllo quando la molteplicità è pari ad h+1. Ciò avviene perchè quanto più aumenta la molteplicità di un nodo x_i tanto più si riduce il supporto della B-Spline definita su quel nodo, e poichè la B-Spline deve avere valori in [0,1] il risultato è che tende sempre più rapidamente al suo valore massimo.



(a) *Molt. 2* (b) *Molt. 3*

Figura 5.5

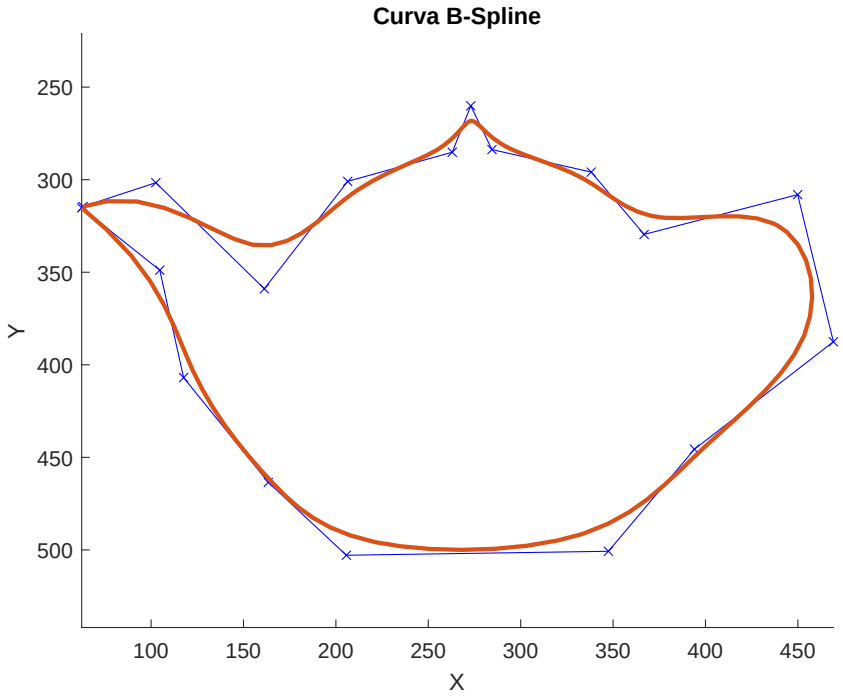
Dalle figure 5.5 si vede chiaramente come all'aumentare della molteplicità del nodo (3,4) la curva B-Spline si avvicina ad esso fino ad interpolarla.

A questo punto non ci resta che rappresentare la Curva B-Spline cubica relativa al nostro caso di studio (*SPLINETEAPOT.m*)

```
load x
load y

P=[x y]
%Seleziono i punti di controllo
controlPoints = P';
%Scelgo l'intervallo temporale selezioando istante iniziale e finale
tInterval = sort(x);%P(:,1);
%Campioni temporali per la traiettoria, specificati come vettore
tSamples = P(:,2);

figure(1);
hold on;
axis('equal');
set(gca, 'YDir','reverse')
[q,~,~,pp]=bsplinepolytraj(controlPoints,tInterval,tSamples)
plot(controlPoints(1,:),controlPoints(2,:),'xb-');
title('Curva B-Spline')
xlabel('X')
ylabel('Y')
hold on
fnplt(pp)
```


Curva B-Spline
250
300
350
400
450
500
Y
100
150
200
250
300
350
400
450
X

Capitolo 6

Curve NURBS

Le curve NURBS sono una descrizione di una classe di curve ampiamente usata nel campo della computer grafica per la definizione dei modelli 3D di qualsiasi forma. Nascono come generalizzazione delle curve B-Spline, le quali sono realizzate attraverso polinomi, caratteristica conveniente in termini di complessità computazionale ma limitante sotto alcuni aspetti:

- Non risulta possibile rappresentare esattamente le curve associate a funzioni trigonometriche, le quali non possono essere rappresentate da un numero finito di polinomi bensì solo da una somma infinita di essi (ovvero tramite sviluppo in serie di Taylor).
- Non risulta possibile rappresentare esattamente tutte le sezioni coniche.

Tuttavia esiste una classe di funzioni polinomiali in grado di descrivere esattamente questo tipo di curve, ovvero i **polinomi razionali**. Ciò, nel 1975, ha portato all'introduzione presso la BOEING Corporation (un'industria aeronautica statunitense produttrice di velivoli per uso civile e militare) di una nuova tipologia di curve B-spline, dette curve NURBS (Non Uniform Rational B-Splines), ottenute generalizzando la definizione già esistente tramite l'impiego di polinomi razionali.

Una curva NURBS è definita da quattro informazioni, di seguito elencate, a partire dalle quali è possibile rappresentare precisamente qualunque forma:

1. Il **Grado**, un intero positivo denotato con la lettera h. Può assumere un valore intero positivo qualsiasi, ma solitamente è variabile nell'intervallo di valori [1,5] per evitare il fenomeno delle oscillazioni. Spesso si fa riferimento, invece che al grado, all'**Ordine**, un intero positivo pari al grado della curva più uno, ovvero h+1.
2. I **punti di controllo**: un insieme di punti di controllo $P_0, ..., P_n$ in numero almeno pari all'ordine della curva NURBS. L'aggiunta di punti di controllo consente una miglior approssimazione ad una data curva.

3. Il **vettore dei nodi**: vettore numerico $T = (t_0, t_1, ..., t_m)$ costituito da m+1 elementi, dove m=n+h+1 ovvero la somma tra il grado della curva ed il numero di punti di controllo. Inoltre se $t_0 = 0$ e $t_m = 1$ il vettore dei nodi è detto "standard".

4. Il **vettore dei pesi**: Le curve NURBS a differenza delle B-Spline hanno un parametro in più che ne modifica la forma, infatti ad ogni punto di controllo è associato un peso $w_i \geq 0 \quad \forall i = 0...n$ che indica la capacità di tale vertice di attrarre a sé la curva. Se i punti di controllo di una curva hanno tutti il medesimo peso ($w_i = w \quad \forall i \in [0, n]$) allora la curva è detta "non razionale", altrimenti "razionale".

Per quanto riguarda il vettore dei nodi, si richiede che vengano soddisfatte diverse "**condizioni tecniche**": il vettore deve essere ordinato in senso crescente e ogni nodo deve avere molteplicità al massimo pari al grado della curva (h). Quest'ultima condizione è dovuta alla proprietà della curva di avvicinarsi ad un dato punto di controllo al crescere della molteplicità di un nodo, fino al massimo di interpolare quel punto quando la molteplicità del nodo diventa pari all'ordine della curva. Tuttavia l'irregolarità della curva cresce al crescere della molteplicità di un nodo, fino a degenerare in una discontinuitià quando la molteplicità del nodo diventa pari esattamente all'ordine. In base a ciò è possibile classificare la molteplicità dei nodi in due categorie: Nodi a **molteplicità piena** (o nodi pieni) se il rispettivo valore si ripete h volte (molteplicità pari al grado della curva) e nodi a **molteplcità semplice** (o nodi semplici), ovvero i nodi a molteplicità unitaria.

In base alle categorie di molteplicità dei nodi è possibile classificare un vettore di nodi:

- **vettore di nodi uniforme**, se la sequenza inizia con un nodo a molteplicità piena, segue con nodi semplici, termina con un nodo a molteplicità piena e tutti i valori sono ugualmente spaziati.
- **vettore di nodi non uniforme** se il vettore non è uniforme.

6.0.1 Formulazione matematica delle curve NURBS

Una volta presentati tutti i parametri che caratterizzano una curva NURBS, possiamo fornire una definizione formale a partire dalla definizione della curva B-Spline precendetemente mostrata aggiungendo opportunamente i pesi. Dati i punti di controllo $P_0...P_n$, i relativi pesi $w_0...w_n$ ed il vettore di nodi $T = (t_0...t_m)_{m=n+h+1}$, la curva NURBS parametrica C(t) si definisce come:

$$C(t) = \frac{\sum_{i=0}^{n} w_i \; P_i \; B_{i,h}(t)}{\sum_{i=0}^{n} w_i \; B_{i,h}(t)} \qquad t \in [t_0, t_{n+h+1}] \tag{6.1}$$

dove con $B_{i,h}(t)$ si indicano le funzioni di base B-Spline di grado h (definite ricorsivamente attraverso la formula di De Boor). É possibile anche utilizzare come definizione una forma compatta equivalente, indicando con $R_{i,h}(t)$ la funzione di base **razionale di grado h** sul vettore dei nodi T e pesi w_i:

$$C(t) = \sum_{i=0}^{n} P_i \; R_{i,h}(t) \tag{6.2}$$

$$R_{i,h}(t) = \frac{w_i \; B_{i,h}(t)}{\sum_{i=0}^{n} w_i \; B_{i,h}(t)} \qquad t \in [t_0, t_{n+h+1}] \tag{6.3}$$

Sostanzialmente aumentando e diminuendo il valore del peso w_i, la curva verrà avvicinata o allontanata sempre più dal punto di controllo P_i (questo perchè aumenterà o diminuirà il corrispettivo valore della $R_{i,h}(t)$). Al tendere del valore del peso w_i all'infinito, la curva passerà esattamente per il punto di controllo P_i.

Per una curva di grado *g*, il peso di qualsiasi punto di controllo è diverso da zero solo in *g+1* intervalli di spazio parametrico. Entro questi intervalli, il peso cambia secondo una funzione polinomiale (funzione base) di grado *g*. Ai confini degli intervalli, le funzioni base si approssimano lentamente a zero, con un valore di smussatura determinato dal grado del polinomio.

6.0.2 Prorpietà delle curve NURBS

Di seguito si elencano alcune delle importanti proprietà di cui godono le curve NURBS:

- **Non negatività**: La funzione $R_{i,h}(t)$ è una funzione razionale di grado h in t, non negativa $\forall i, \forall h$ e $\forall t$ ed in particolare assume valori non nulli solo per $t \in [t_i, t_{i+h+1}]$ (**proprietà del supporto locale**).

- **Partizionamento dell'unità**: $\sum_{i=0}^{n} B_{i,h} = 1$, ovvero la somma di tutte le funzioni base $B_{i,h}(t)$ su ogni intervallo $[t_i, t_{i+h+1}]$ è pari ad 1.

- **Differenziabilità**: Tra i nodi è C^{∞}, in corrispondenza di nodi multipli con molteplicità pari a k, la curva NURBS C(t) è una funzione di classe C^{h-k} (infatti all'aumentare della molteplicità di un nodo decresce il livello di continuità nella curva, mentre cresce all'aumentare del grado h).

- **Strong convex hull**: La curva NURBS C(t) è interamente contenuta nel dominio convesso determinato a partire dai suoi punti di controllo se e solo se $w_i \geq 0 \; \forall i$.

- **Invarianza alle trasformazioni affini**: Definita innanzitutto una trasformazione affine come una trasformazione lineare seguita da una traslazione, tale proprietà garantisce il fatto che possiamo applicare la trasformazione affine ai punti di controllo e la curva NURBS trasformata sarà quella definita dai punti di controllo trasformati.

- **Rappresentazione di sezioni coniche**: una importante proprietà delle NURBS è quella di poter essere utilizzate per rappresentare una qualunque sezione conica. In particolare data una

curva NURBS di grado 2, con vettore dei noti T = (0,0,0,1,1,1) e tre punti di controllo $P_0, P_1 e P_2$, lavorando unicamente sul peso w_i (avendo posto $w_0 = w_2 = 1$) possiamo ottenere:

- una **parabola** se $w_i = 1$
- un'**ellisse** se $w_i < 1$
- una **iperbole** se $w_i > 1$
- una **retta** se $w_i = 0$

Di seguito si mostra uno script matlab (*NURBS_ coniche.m*) ed il corrispettivo output (6.1) che mostra un esempio di rappresentazione delle coniche appena elencate (compresa la retta che è una conica degenere)

```
%Esempio plotting coniche con NURBS
clear all; close all;
h = 2;
t = [0 0 0 1 1 1];
P = [0 2 4;1 3 1];
U = linspace(0,1,1e4);
w1 = [1 1 1];
c = Funzione_Nurbs(P,h+1,t,w1);
plot(c(1,:),c(2,:),'r')
hold on
w2 = [1 0.5 1];
c = Funzione_Nurbs(P,h+1,t,w2);
plot(c(1,:),c(2,:),'g')
w3 = [1 2 1];
c = Funzione_Nurbs(P,h+1,t,w3);
plot(c(1,:),c(2,:),'y')
w4 = [1 0 1];
c = Funzione_Nurbs(P,h+1,t,w4);
plot(c(1,:),c(2,:),'m')
plot(P(1,:),P(2,:),'b')
scatter(P(1,:),P(2,:),'bo','filled')
legend('parabola(w_1=1)','ellisse(w_1<1)','iperbole(w_1>1)','retta(w_1=0)')
axis([-1 5 0 4])
```

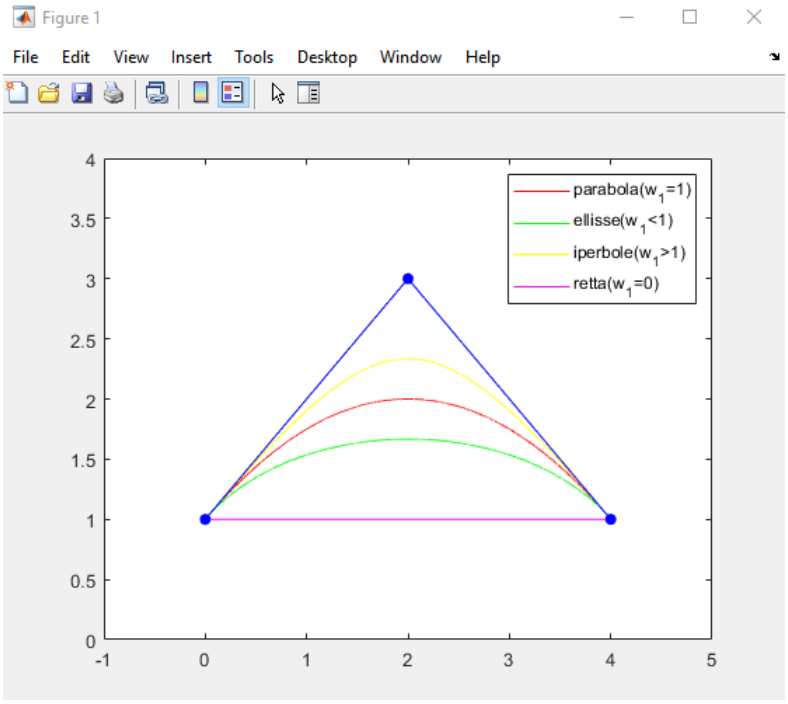


Figura 6.1

6.0.3 Rappresentazione di una circonferenza

A titolo esemplificativo si mostra un confronto tra una B-Spline ed una NURBS (con pesi e vettore dei nodi di controllo opportunamente impostati) nella approssimazione di una circonferenza: si può notare come la curva NURBS risulti decisamente migliore nella descrizione della circonferenza (la quale è una conica) rispetto alla B-Spline.

Nella prima parte dello script (*Circonferenza_Nurbs_e_BSpline.m*) avviene la fase di setting dei parametri della NURBS con relativa generazione della curva e plotting. In particolare sono definibili dall'utente i punti di controllo, l'ordine della curva, vettore dei nodi e vettore dei pesi. Notiamo che il vettore dei nodi è stato impostato in maniera tale che primo ed ultimo punto abbiano molteplicità piena, in modo tale che la curva passi per essi, punti che tra l'altro sono coincidenti (P_1=P_7=(1,2)). Si noti inoltre che il vettore dei pesi è stato impostato in modo tale da dare un maggior peso attrattivo ai punti di controllo sul diametro ed un peso inferiore agli altri punti.

```
close all, clear all; %pulizia workspace e command window
Punti_Controllo=[1 1 3 3 3 1 1;2 3 3 2 1 1 2]; %definizione punti di controllo
Numero_Punti_Controllo=size(Punti_Controllo,2); %numero di punti di controllo
Vettore_Nodi=[0,0,0,0.25,0.5,0.5,0.75,1,1,1]; %definizione vettore dei nodi
Vettore_Pesi=[1 0.5 0.5 1 0.5 0.5 1]; %definizione vettore dei pesi
Ordine_Curva=3; %definizione ordine della curva

%PLOTTING CURVA NURBS
%NOTA: i punti di controllo sono (1,2)(1,3)(3,3)(3,2)(3,1)(1,1)(1,2)
```

```
%primo ed ultimo punto coincidenti (ma pesi differenti), è stato imposto
%il passaggio per il primo e l'ultimo punto aumentando la molteplicità dei
%rispettivi nodi.
Curva_Nurbs=Funzione_Nurbs(Punti_Controllo,Ordine_Curva,Vettore_Nodi,Vettore_Pesi);
Plotting_NURBS=plot(Curva_Nurbs(1,:),Curva_Nurbs(2,:),'LineWidth',1);
set(Plotting_NURBS,'Color','blue');
hold on;

%PLOTTING CURVA B-SPLINE
%Una curva B-spline può essere ottenuta da una NURBS imponendo tutti i pesi ad 1
Vettore_Pesi_Spline=[1 1 1 1 1 1 1];
Vettore_Nodi_Spline=[0:1/(Numero_Punti_Controllo+Ordine_Curva):1];
Curva_Spline=Funzione_Nurbs(Punti_Controllo,Ordine_Curva,Vettore_Nodi_Spline,Vettore_Pesi_Spline);
Plotting_SPLINE=plot(Curva_Spline(1,:),Curva_Spline(2,:),'LineWidth',1);
set(Plotting_SPLINE,'Color','red');
hold on;

%PLOTTING PUNTI DI CONTROLLO
plot(Punti_Controllo(1,:),Punti_Controllo(2,:),'-O','MarkerFaceColor', 'yellow');
legend('NURBS','B-Spline','Punti di Controllo');
```

Successivamente è stata costruita, a partire dalla funzione di costruzione di una curva NURBS, e plottata la corrispettiva curva B-Spline. Ciò è stato fatto imponendo i pesi ad un valore unitario. In figura 6.3 si mostra l'output di tale script.

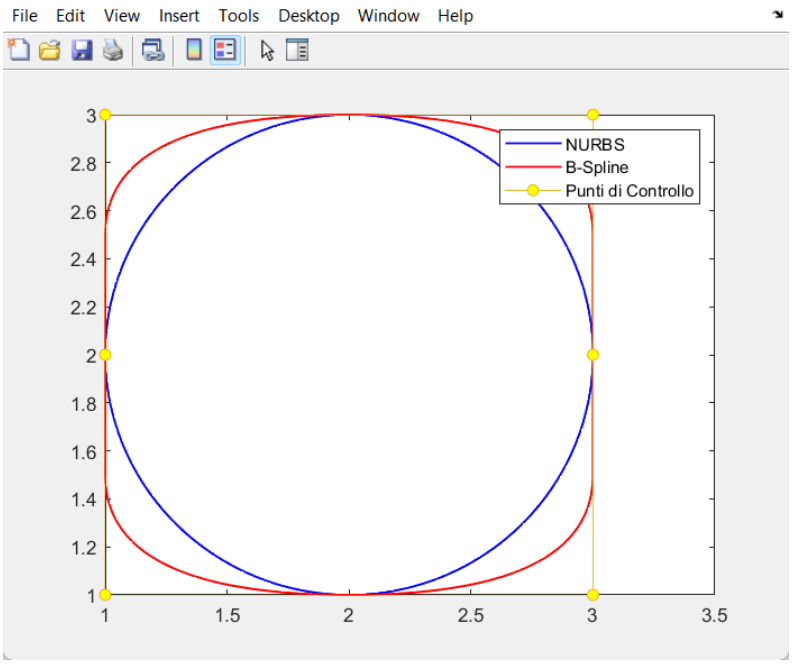


Figura 6.2

6.0.4 Rappresentazione teapot con curva NURBS

Uno dei principali motivi dell'ampia popolarità delle NURBS è che offrono una forma matematica comune per rappresentare con precisione sia forme analitiche standard sia curve a forma libera, offrendo alta flessibilità nella progettazione di un'ampia varietà di forme manipolando opportunamente i suoi parametri. Tuttavia presentano anche degli aspetti critici tra i quali si cita la necessità di avere un maggiore spazio di archiviazione per rappresentare curve e superfici tradizionali, oppure la facilità con cui si può incombere in una pessima parametrizzazione e formulazione della curva stessa dettata ad esempio da una scelta errata dei pesi. Si presenta difatti di seguito un esempio di errata parametrizzazione nella rappresentazione del profilo di una teapot con curva NURBS di grado 5, nel quale risulta evidente come con la curva B-spline si sia ottenuto un risultato maggiormente soddisfacente.

```
%Teapot_Nurbs_Plotting.m
close all, clear all;

%%PLOTTING CURVA NURBS ORDINE 5
load y_teapot;
load x_teapot;
Punti_Controllo=[x';y']; %definizione punti di controllo
Numero_Punti_Controllo=size(Punti_Controllo,2); %numero di punti di controllo
Vettore_Nodi=[0,0,0, 0.25,0.25,0.25, 0.35,0.35,0.35, 0.5,0.5,
              0.5, 0.6,0.6, 0.75,0.75,0.75, 0.8,0.8,0.8, 1,1,1];
Vettore_Pesi=[1 1 0.80 1 0.5 0.6 0.9 1 1 0.6 0.6
              0.95 0.93 0.55 0.9 0.3 1 1]; %definizione vettore dei pesi
Ordine_Curva=5;
Curva_Nurbs =  Funzione_Nurbs(Punti_Controllo,Ordine_Curva,Vettore_Nodi,Vettore_Pesi);
axis('equal');
set(gca,'YDir','reverse')
Plotting_NURBS=plot(Curva_Nurbs(1,:),Curva_Nurbs(2,:),'LineWidth',1);
set(Plotting_NURBS,'Color','blue');
hold on;

%PLOTTING PUNTI DI CONTROLLO
axis('equal');
set(gca,'YDir','reverse')
plot(Punti_Controllo(1,:),Punti_Controllo(2,:),'-O','MarkerFaceColor', 'yellow');
legend('NURBS','Punti di Controllo');
```

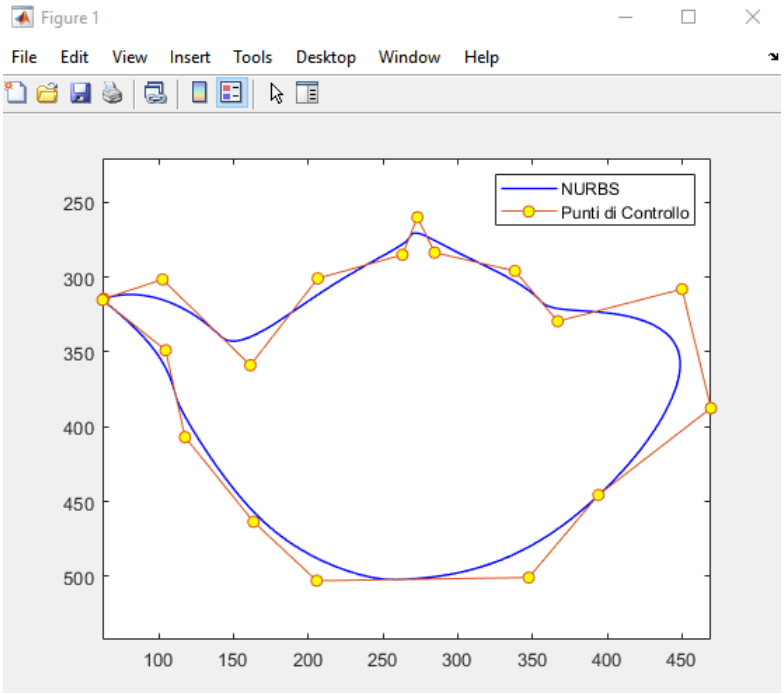


Figura 6.3

6.1 Confronto curve di Bézier, B-Spline e NURBS

In conclusione si riporta (in figura 6.4) un confronto sintetico e tabellare tra le tipologie di curve trattate fino ad ora.

	BEZIER	B-SPLINE	NURBS
La curva è contenuta all'interno dell'involucro convesso	SI	SI	SI (se il peso dei punti del poligono di controllo è positivo)
Legame tra il grado della curva e il poligono di controllo	Grado = numero di vertici - 1	NO	NO
La curva tocca il poligono di controllo?	SI, nel primo e nell'ultimo punto	NO (ma può essere fatto collassando i nodi al grado della curva)	NO (ma può essere fatto collassando i nodi al grado della curva)
La forma della curva e del poligono sono correlate?	NO, solo in modo molto approssimato	SI, e li si può far coincidere	SI, e li si può far coincidere
Si può avere il controllo locale sulla curva?	NO	SI	SI
Si possono disegnare circonferenze ed ellissi?	NO	NO	SI

Figura 6.4

Capitolo 7

Superfici di Bézier

Le superfici di Bézier sono un potente strumento per la progettazione di superfici, ampiamente utilizzate nella computer graphics assieme alle curve di Bézier. Ancora una volta, l'idea alla base è la definizione di un numero finito di punti di controllo nello spazio per costruire il cosiddetto **poliedro di controllo**.

Definizione:

Sia [a,b]x[a,b]=[0,1]x[0,1] e si fissino (k+1)(h+1) punti in $P_{0,0}, ..., P_{k,k} \in \mathbb{R}^3$. Una Superficie di Bézier è definita dalla seguente equazione

$$Q(u,v) = \sum_{i=0}^{h} \sum_{j=0}^{k} B_{i,k}(u) B_{j,k}(v) P_{i,j} \tag{7.1}$$

Dove $B_{i,h}(u)$ e $B_{j,k}(v)$ sono polinomi di Berstein.

Le superfici di Bézier presentano però una serie di limitazioni:

- Non è possibile rappresentare sfere nè superfici di rotazione
- Non ammettono controllo locale
- Per rappresentare forme complesse, usando una sola superficie, sono necessari molti punti di controllo. Questo significa che il grado dei polinomi usati nella rappresentazione di Bézier può diventare anche molto alto e quindi difficile da trattare al calcolatore.

E' chiaro che il problema di elaborazione di una matrice di punti di controllo di grandi dimensioni è intrattabile. La soluzione a questo problema consiste nell'"incollare" più superfici di grado minore secondo condizioni di continuità per dare l'effetto di una sola superficie, la cui elaborazione risulta però meno complessa. Si tratta delle cosiddette **patches di Bézier**.

7.1 Cubic Bézier Surface e Teapot

Un superficie di Bézier cubica è costituita da una o più Bézier patches. Una Patch di Bézier è una funzione di due variabili con un array di punti di controllo. Un patch cubica di Bézier è costruita a partire da una array 4x4 di punti di controllo e parametrizzata a partire da due variabili u e v fornite dall'equazione:

$$Q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_{i,3}(u)B_{j,3}(v)P_{i,j} \qquad 0 \leq u,v \leq 1 \tag{7.2}$$

con $B_{i,3}(u)$ e $B_{j,3}(v)$ impiegati come funzioni che saldano i punti di controllo assieme).

Esistono, in realtà, diverse condizioni di incollaggio che hanno la funzione di rendere la superficie più o meno levigata, in particolare:

- La continuità C^0 impone che le due superfici condividano lo stesso bordo, quindi in prossimità dell'incollamento è possibile avere punti angolosi. Ad esempio considerando i bodi v=1 del patch P1 e v=0 del patch P2 la condizione diventa

$$P_1(u,1) = P_2(u,0) \qquad per \quad ogni \quad u \in [0,1] \tag{7.3}$$

- La continuità C^1 impone, oltre alla C^0, che la derivata prima e seconda lungo il bordo sia la stessa eliminando la presenza di punti angolosi:

$$\frac{\partial P_1}{\partial v}(u,1) = \frac{\partial P_2}{\partial v}(u,0) \tag{7.4}$$

$$\frac{\partial^2 P_1}{\partial^2 v}(u,1) = \frac{\partial^2 P_2}{\partial^2}(u,0) \tag{7.5}$$

Nel nostro caso di studio la superficie della Teapot è costruita utilizzando 32 patches cubiche di Bézier. L'equazione della singola patch viene riportata di seguito:

$$Q(u,v) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}$$

In particolare, carichiamo la matrice S la quale contiene tutti i punti di controllo di tutte le patches della superficie considerata. S(:,:,:,k) presenta i punti di controllo della k-esima patch (con k=1...32). La dimensione di S(:,:,:,k) è 4x4x3 ovvero abbiamo 16 punti di controllo ognuno dei quali è caratterizzato da 3 coordinate (x,y,z) rispettivamente contenuti in S(:,:,1,k) , S(:,:,2,k), S(:,:,3,k)

Di seguito riportiamo il codice relativo al plotting in 3D della superficie considerata:

```
clc, close all, clear all

load('teapot');
[r c d np]=size(S);
% % np: numero di patches
ni=20; %number of interpolated values between end control points
u=linspace(0,1,ni); v=u;
% %Più eleveto è il valore di ni smoother è la superficie
%ma maggiore è il calcolo computazionale
% % -------------------------------------
% % Cubic Bezier interpolation of control points of each patch
for k=1:np
    Q(:,:,:,k)=bezierpatchinterp(S(:,:,:,k),u,v); %interpolation of kth patch
end
plotbeziersurface3D(S,Q)
```

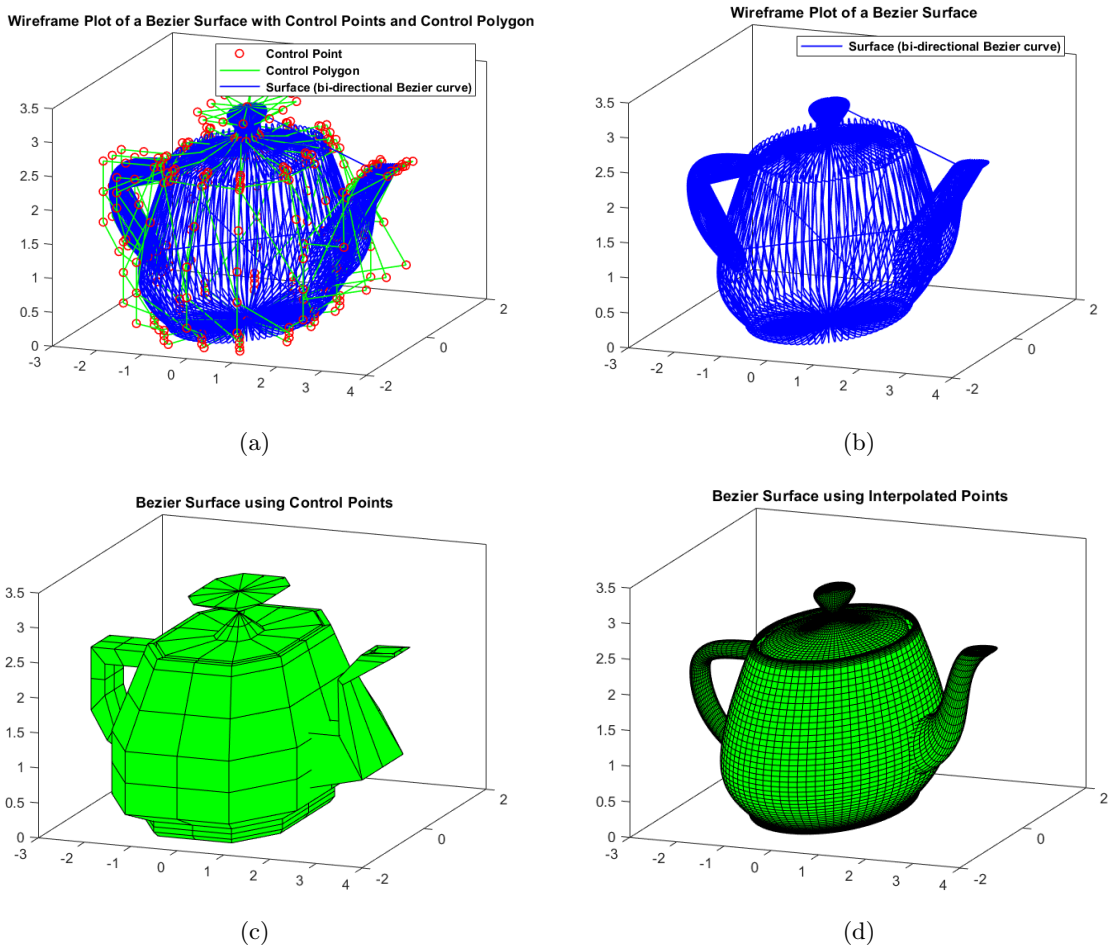


Figura 7.1: Teapot Bézier Surface