Fitting dei dati

Dall'interpolazione polinomiale alle curve NURBS

Martedì Gaetano M63/1226
Salzillo Biagio M63/1227
A.A. 2020/21

Sommario

CAPITOLO 1

1 Introduzione

Per comprendere al meglio la natura di un fenomeno, non basta soltanto raccogliere buoni dati sperimentali: occorre anche saperli interpretare. Ciò che si cerca di fare è trovare una funzione matematica che approssimi al meglio i dati raccolti. Il processo di costruzione di una curva o di una funzione matematica, che meglio "spieghi" i suddetti dati, prende il nome di *fitting*.

Nel corso di questo elaborato verranno illustrate varie tecniche di fitting dei dati, applicate ad un caso studio reale, ovvero al set di dati relativo alla temperatura media della città di Milano per un arco temporale di 22 anni: dal 1984 al 2006. I dati sono stati reperiti dal sito Web dell'ente statunitense National Centers for Environmental Information (NCEI).

Il problema del fitting consiste nel costruire un corretto modello matematico che descriva i dati in modo attendibile; tale rappresentazione può avvenire attraverso due modelli: uno interpolante e uno approssimante. Con il primo modello, si costruisce una curva che passa per i punti assegnati, trascurando l'eventuale errore presente sui dati, mentre con il secondo, si costruisce una curva che non passa necessariamente per i punti dati, considerando questi ultimi affetti da errore.

1.1 Interpolazione

Problema generale di INTERPOLAZIONE

Dati n punti distinti $(x_i\ , y_i)_{i=1,..,n}$ si vuole costruire una funzione $f(x)$ tale che nei nodi $(x_i)_{i=1,..,n}$ soddisfi certe condizioni, dette condizioni interpolanti.

Interpolazione di Lagrange

Sia F uno spazio di funzioni di variabile reale o complessa.

Assegnati:

- n valori distinti reali o complessi $\{x_i\}_{i=1,..,n}$
- n valori distinti reali o complessi $\{y_i\}_{i=1,..,n}$

Si cerca una funzione $f(x) \in F$ tale che:

$$f(x_i) = y_i \qquad i = 1, \dots, n \qquad (condizioni\ di\ interpolazione\ di\ Lagrange)$$

ovvero tali condizioni si traducono col fatto che la funzione deve passare per i punti assegnati.

Interpolazione di Hermite

Data una funzione $f(x)$ di cui, nei punti x_i , per $i = 1, \dots, n$, si conoscono i valori $y_i = f(x_i)$ e i valori delle sue derivate fino all'ordine q, con l'interpolazione di Hermite si cerca una funzione interpolante che, nei punti x_i, assuma gli stessi valori della funzione $f(x)$ originale e delle sue derivate, fino all'ordine q.

1.2 Approssimazione

Problema generale di APPROSSIMAZIONE

Dati n punti distinti $(x_i , y_i)_{i=1,..,n}$ si vuole costruire una funzione $f(x)$ tale che nei nodi $(x_i)_{i=1,..,n}$ non assuma i valori $(y_i)_{i=1,..,n}$ ma si scosti poco da essi.

L'idea principale, per la costruzione di una curva approssimante, è quella di unire coppie di punti mediante curve piane che però abbiano una traiettoria "controllata" da alcuni punti intermedi, detti **punti di controllo**, uniti in una poligonale chiamata "**convex hull**". Questo è possibile imponendo condizioni sui punti di passaggio e sulla velocità e curvatura della traiettoria che si intende seguire (quindi bisogna introdurre un controllo sulle derivate nei punti). Le curve di Bézier sono un classico strumento matematico per la costruzione di curve algebriche approssimanti a partire dai punti di controllo. Un metodo numericamente stabile per calcolare le curve di Bézier è l'algoritmo di De Casteljau.

CAPITOLO 2

2 Interpolazione polinomiale

2.1 Il metodo dei minimi quadrati

Siano (x_i, y_i) con $i = 1, 2, \dots, n$ i punti che rappresentano i dati in ingresso. Si vuole trovare una funzione f tale che approssimi la successione di punti dati. Questa funzione può essere determinata minimizzando la distanza (euclidea) tra le due successioni y_i e $f(x_i)$, ovvero la quantità d:

$$d = \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

da cui il nome "minimi quadrati".

Nei casi pratici, in genere, $f(x)$ è parametrica: in questo modo il problema si riduce a determinare i parametri che minimizzano la distanza dei punti dalla curva. Naturalmente, per ottenere un'unica curva ottimizzata, e non un fascio di curve, è necessario possedere un numero di punti sperimentali maggiore del numero di parametri da cui dipende la curva; in questi casi il problema è detto *sovradeterminato*.

Nel caso dell'interpolazione polinomiale, la funzione $f(x)$, oltre ad essere parametrica, deve avere la forma di un polinomio per l'appunto, i cui coefficienti saranno proprio i parametri da determinare per minimizzare la distanza d. Il grado r del polinomio cercato può essere fissato a priori, purché sia pari almeno al numero di punti da interpolare (x_i, y_i) meno 1, cioè $r < n - 1$. Questo perché un polinomio di grado r possiede $r + 1$ coefficienti da determinare dunque, per avere un problema dei minimi quadrati sovradeterminato dati n punti di controllo, sono necessari almeno n coefficienti del polinomio.

Esempio del problema dei minimi quadrati con polinomio interpolante di primo grado:

$$argmin_{(a_0, a_1)} d$$

$$d = \sum_{i=1}^{n} [y_i - (a_0 + a_1 x_i)]^2$$

2.2 Polyfit e polyval

La funzione *polyfit* in MATLAB ha esattamente come scopo quello di determinare i coefficienti del polinomio interpolante, nel senso dei minimi quadrati, dati in input i punti $(x_i, y_i)_{i=1,..,n}$ da interpolare ed il grado r del polinomio cercato.

```
>> 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).
```

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)
```

Vediamo un esempio di interpolazione polinomiale, attraverso le funzioni *polyfit* e *polyval*, applicate al caso studio sulla temperatura media annuale della città di Milano. Riportiamo lo script file di MATLAB *interp_polinomiale.m* :

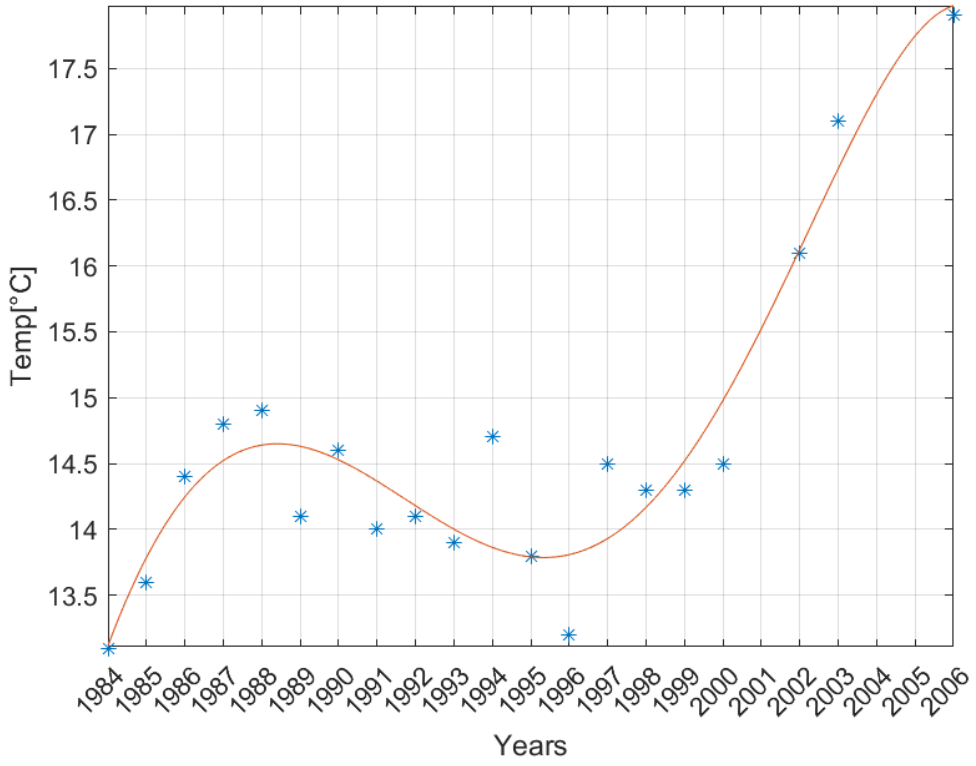
```
load tavg_milan_v2
[v,t] = interp_pol(years,temp,5);
plot(years,temp,'*',t,v)
grid on
axis tight
xticks(min(years):max(years))
xlabel('Years')
ylabel('Temp[°C]')
clear
```

Che a sua volta fa uso della function MATLAB *interp_pol.m*:

```
function [v,t] = interp_pol(x,y,r)
p = polyfit(x,y,r);
```

```
t = min(x):0.1:max(x);
v = polyval(p,t);
end
```

Il grafico risultante è il seguente:



CAPITOLO 3

3 Curve di Bézier

Come già accennato in precedenza, le curve di Bézier sono delle particolari curve parametriche. Una curva di Bézier è il centro di massa del convex hull. Il centro di massa (CM) è il punto di un oggetto in cui è concentrato il suo peso, ovvero dove si può assumere che la forza di gravità sia applicata. Per un insieme finito di n punti può essere definito come:

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

In particolare, nelle curve di Bézier come posizioni P_i si prendono i punti di controllo. Ad esempio, l'equazione della curva (o centro di massa) per quattro punti di controllo è:

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

Per stabilire chi svolge il ruolo delle masse si analizzano le proprietà richieste alle curve di Bézier.

Proprietà delle curve

1. La curva deve interpolare il primo punto P_0 e l'ultimo punto P_n;
2. Tutti i punti della curva devono stare all'interno del convex hull;
3. La somma CM deve definire un punto del piano;
4. La curva deve essere indipendente dal sistema di riferimento cartesiano (cioè cambiando sistema ottengo sempre la stessa forma della curva).

Da queste prime proprietà si arriva alla conclusione che la somma delle m_i deve essere 1 e quindi l'equazione della curva di Bézier assume la forma:

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

Le curve di Bézier vengono rappresentate da un polinomio di grado pari al numero di punti di controllo meno uno.

Per stabilire chi siano le m_i si analizzano altre proprietà delle curve relative alle derivate di diverso ordine calcolate nei punti di controllo. In conclusione, dati $n + 1$ punti di controllo, la curva di Bézier di grado n è definita conoscendo la derivata:

- zero in P_0
- zero in P_n
- prima in P_0
- prima in P_n
- seconda in P_0
- seconda in P_n

  …
- derivata di ordine $n - 1$ in P_0
- derivata di ordine $n - 1$ in P_n

Tutte queste condizioni permettono la costruzione di $n + 1$ 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.

In conclusione, Bézier definì le funzioni m_i utilizzando i polinomi di Bernstein, ovvero polinomi che fanno sì che la curva soddisfi le proprietà sopracitate. L'equazione della curva di Bézier è così fatta:

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

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

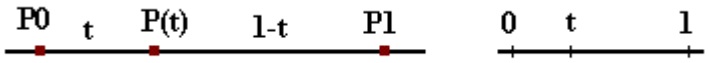
$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad t \in [0,1]$$

3.1 Algoritmo di De Casteljau

L'algoritmo di De Casteljau è un metodo che permette di costruire in modo semplice e algoritmico le curve di Bézier, aventi rappresentazione parametrica di tipo polinomiale. Fissato un valore $t \in [0,1]$, esso permette di calcolare il punto corrispondente sulla curva $C(t)$ mediante interpolazioni lineari ripetute a partire dai punti di controllo. Come già detto in precedenza, il grado della curva è determinato dal numero di vertici del convex hull, procediamo quindi per gradi.

Curva di grado 1

Siano P_0 e P_1 due punti del piano rappresentati come segue, con $t \in [0,1]$:



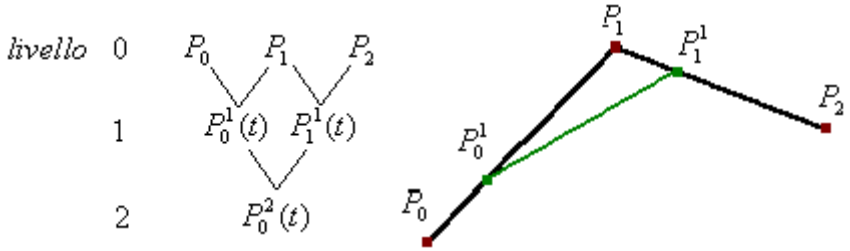
Allora un punto qualunque della curva $C(t)$ compreso tra P_0 e P_1 sarà:

$$C(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1$$

Curva di grado 2

Con tre punti P_0, P_1, P_2 creiamo due curve ausiliarie di grado 1, date dai segmenti $P_0\ P_1$ e P_1P_2. Considerato un valore t in [0,1], si definiscono due punti sui segmenti $P_0\ P_1$ e P_1P_2, unendoli otteniamo un altro segmento chiamato poligonale di primo livello.

Sul nuovo segmento si individua un terzo punto in funzione di t, ovvero quello corrispondente alla curva di Bézier di grado due, relativo ad un determinato valore di t. Ripetendo questo algoritmo per ogni $t \in [0,1]$, ottengo tutti i punti della curva.



L'equazione generale dei punti della curva è quindi ottenuta per sostituzione come segue:

$$P_0^1(t) = (1-t)P_0^0 + tP_1^0$$

$$P_1^1(t) = (1-t)P_1^0 + tP_2^0$$

$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t)$$

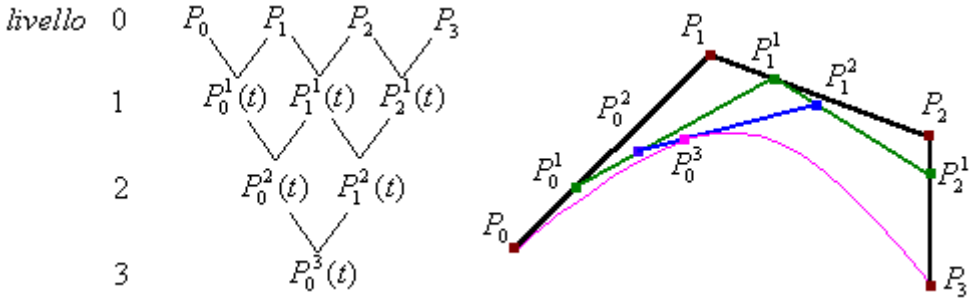
Sostituendo i valori di $P_0^1(t)$ *e* $P_1^1(t)$ otteniamo l'equazione della curva di Bézier

$$C(t) = P_0^2(t) = (1-t)^2P_0^0 + 2(1-t)tP_1^0 + t^2P_2^0$$

Quindi la forma di una curva intermedia in generale è data da:

$$P_j^r(t) = (1-t)P_j^{r-1}(t) + tP_{j+1}^{r-1}(t)$$

Curva di grado 3



$$C(t) = P_0^3(t) = (1-t)^3P_0^0 + 3(1-t)^2tP_1^0 + 3(1-t)t^2P_2^0 + t^3P_3^0$$

La complessità di questo algoritmo è $O\ (n^2)$ per ogni punto della curva, dato che vi sono due cicli innestati sul vettore dei punti di controllo che viene passato in ingresso.

L'algoritmo si presenta:

- Stabile: i coefficienti $1\ -\ t$ e t sono compresi tra 0 e 1 e quindi non c'è amplificazione dell'errore.
- Ricorsivo: viene applicata la stessa formula per ogni passaggio.

3.2 Applicazione in MATLAB

In questo paragrafo, verrà presentato il codice MATLAB per la costruzione della curva di Bézier del caso studio in esame. MATLAB mette a disposizione una funzione che permette di generare i polinomi di Bernstein, ovvero:

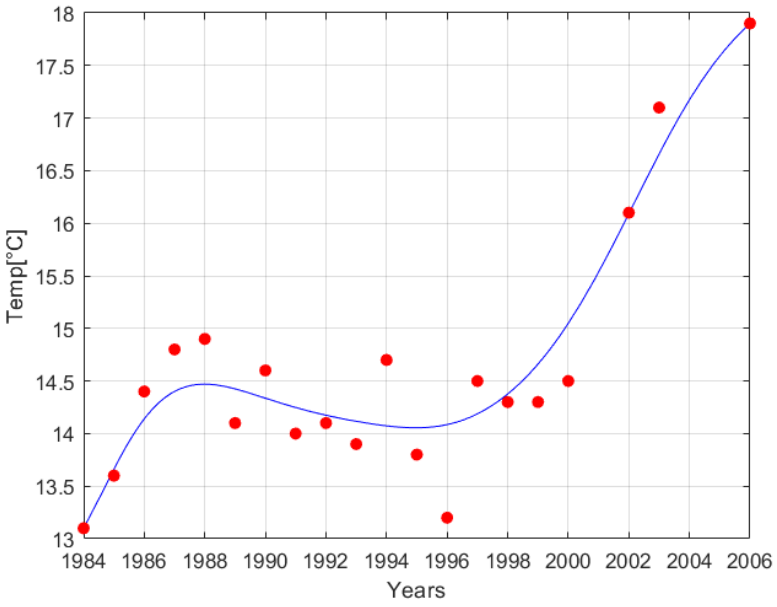
```
B = bernsteinMatrix(k,t)
```

Tale funzione restituisce un vettore n-dimensionale, contenente gli n polinomi di Bernstein di grado $k = n - 1$ (dove n è il numero dei punti di controllo).

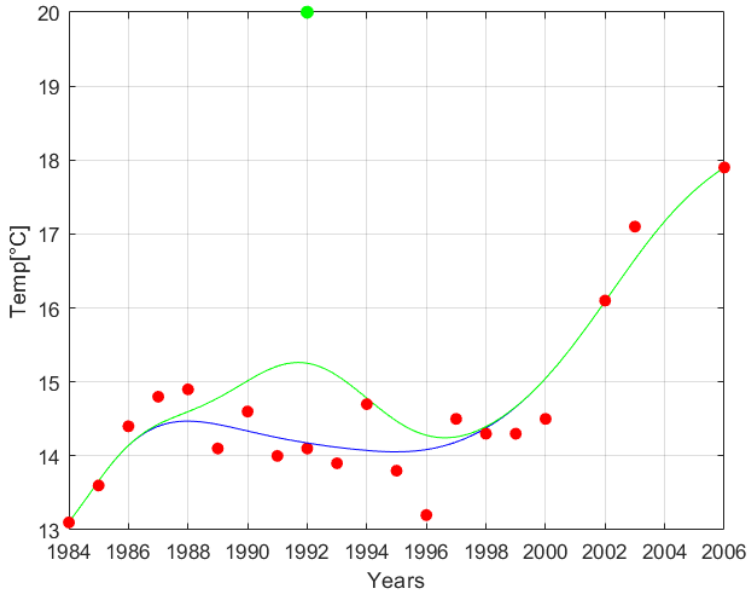
Di seguito il codice MATLAB per costruire e graficare la curva di Bézier:

```
%Calcolo della matrice di Bernstein
syms t
B = bernsteinMatrix(length(years)-1,t)
BézierCurve = simplify(B*tavg)
fplot(BézierCurve(1),BézierCurve(2),[0,1])
hold on
scatter(tavg(:,1),tavg(:,2),'filled')
```

Infine, viene presentato il grafico della curva di Bézier del caso studio di riferimento:



Proviamo adesso a cambiare un solo punto di controllo (P_{1992}), in particolare quello relativo all'anno 1992. Graficando la nuova curva di Bézier (dalla figura, la curva in verde) osserviamo che l'andamento è cambiato, rispetto alla precedente curva. Quindi ciò che si osserva è che non si possono apportare modifiche "locali" alla curva, infatti modificando un solo punto di controllo viene modificata la curva "globalmente", questo perché le funzioni di base (polinomi di Bernstein) sono definite e sono non nulle in tutto l'intervallo di definizione della curva. Per ovviare a questo problema si può utilizzare un modello interpolante o approssimante a tratti.

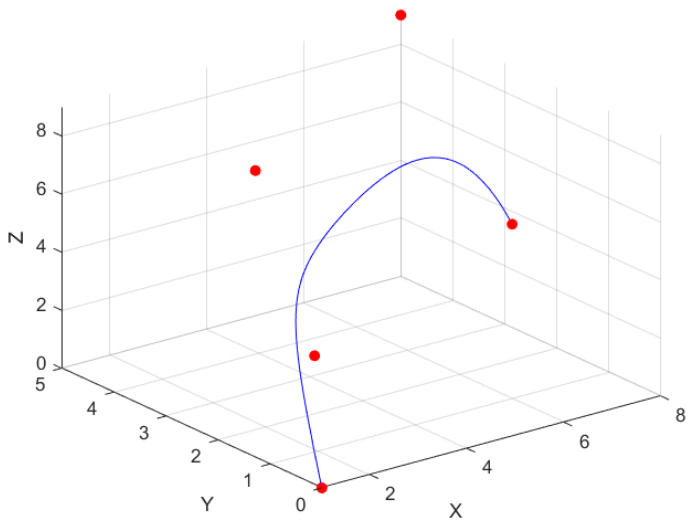


3.3 Curve di Bézier 3D

Presenteremo in questo paragrafo, in maniera analoga al precedete, la costruzione delle curve di Bézier in tre dimensioni tramite MATLAB.

```
%Matrice costruita a partire dai punti di controllo
P = [1 0 0; 5 5 5; 3 2 2; 8 5 9; 6 1 6]
%Calcolo della matrice di Bernstein
syms t
B = bernsteinMatrix(length(P)-1,t)
BézierCurve = simplify(B*P)

fplot3(BézierCurve(1),BézierCurve(2),BézierCurve(3),[0,1],'b')
xlabel('X')
ylabel('Y')
zlabel('Z')
hold on
scatter3(P(:,1),P(:,2),P(:,3),'filled','r')
grid on
```

Di seguito la figura che rappresenta la curva di Bézier in tre dimensioni dati cinque punti di controllo.



3.4 Il fenomeno di Runge

Sia nel caso dell'interpolazione polinomiale col metodo dei minimi quadrati, che nel caso dell'approssimazione con le curve di Bézier, abbiamo utilizzato i polinomi come funzioni interpolanti. Tuttavia, l'impiego di tali funzioni per l'interpolazione di dati comporta uno svantaggio: i polinomi presentano forti oscillazioni al crescere del loro grado. Questa proprietà è una forte limitazione per l'impiego dei polinomi nei processi d'interpolazione, dato che il grado delle curve di Bézier dev'essere pari al numero di punti di controllo meno uno. Quindi al crescere del numero di punti di controllo, cioè al crescere dei dati che disponiamo di un fenomeno, l'errore d'interpolazione cresce, a causa dell'instabilità dei polinomi. Tale problema risulta evidente osservando il *fenomeno di Runge,* dal nome del matematico che ha studiato tale casistica.

Consideriamo la funzione:

$$f(x) = \frac{1}{1+25x^2}$$

Runge scoprì che interpolando questa funzione su un insieme di punti x_i, con $i = 1, \dots, n$, equidistanti nell'intervallo $[-1,1]$, con un polinomio $P_r(x)$ di grado $r \leq n$, l'interpolazione risultante oscilla in ampiezza verso gli estremi dell'intervallo (in questo caso -1 e $+1$).

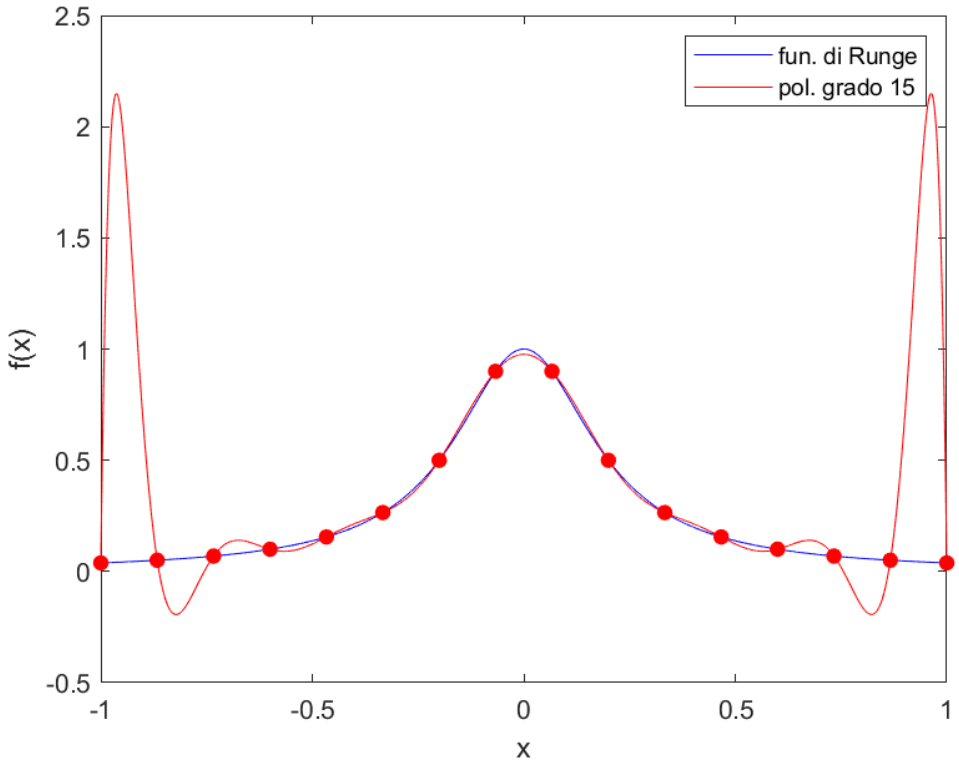
Inoltre, si dimostra che l'errore tra $f(x)$ e $P_n(x)$ tende all'infinito all'aumentare del grado del polinomio:

$$\lim_{n\to\infty} \left(\max_{x\in[-1,1]} |f(x) - P_n(x)| \right) = +\infty$$

Nel file MATLAB *runge.m* grafichiamo l'andamento della funzione $f(x)$ tra -1 e 1, e della curva del polinomio interpolante, nel senso dei minimi quadrati, di grado 15, per mostrare le oscillazioni di quest'ultima:

```
runge = @(x) 1./(1+25*x.^2);
x = linspace(-1,1,1e4);
y = runge(x);
xp = linspace(-1,1,16);
yp = runge(xp);
[v,t] = interp_pol(xp,yp,length(xp)-1);
figure(1)
plot(x,y,'b',t,v,'r')
```

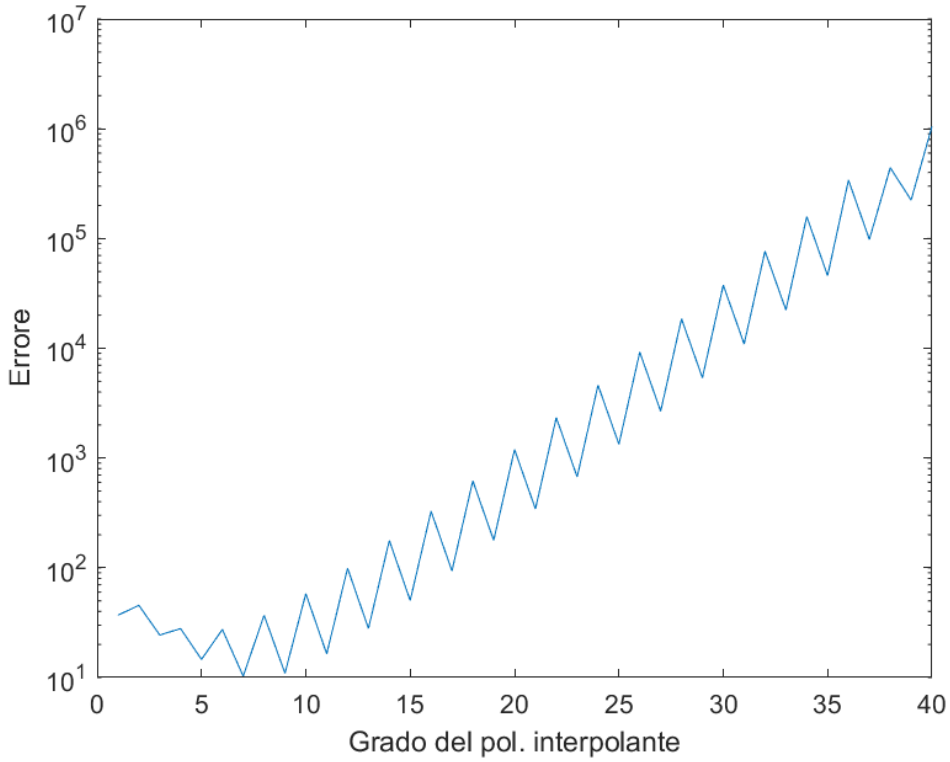
```
hold on
scatter(xp,yp,'r','filled')
xlabel('x');
ylabel('f(x)')
legend('fun. di Runge','pol. grado 15')
hold off
```



Inoltre, mostriamo anche l'andamento dell'errore di approssimazione al crescere del grado della curva interpolante:

```
err = zeros(50,1);
for i = 1:50
    xp = linspace(-1,1,i+1);
    yp = runge(xp);
    [v,t] = interp_pol(xp,yp,i);
    y = runge(t);
    err(i) = norm(y-v,Inf);
end
figure(2)
semilogy(err);
```

```
xlabel('Grado del pol. interpolante')
ylabel('Errore')
```



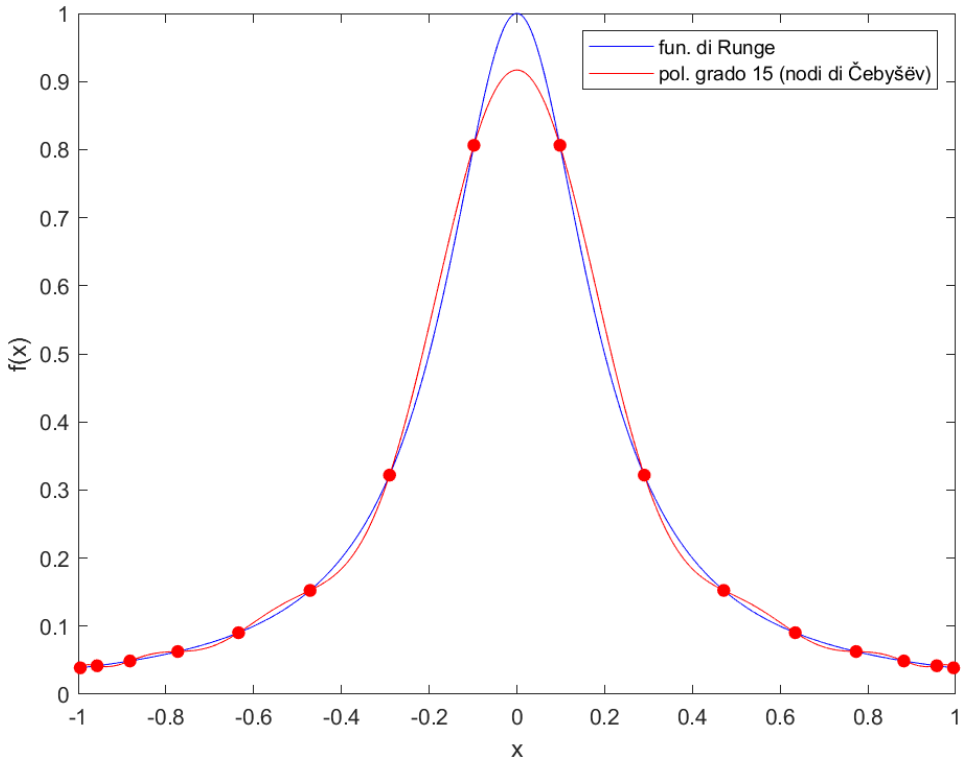
Tuttavia, è possibile ottenere uno schema di interpolazione il cui errore diminuisca all'aumentare del numero di nodi considerati, utilizzando i **nodi di Čebyšëv** in alternativa ai punti equispaziati nell'intervallo $[-1,1]$. I nodi di Čebyšëv sono le radici dei polinomi di Čebyšëv, ossia per ogni $n \in \mathbb{N}$, il polinomio n-esimo possiede n radici semplici interne all'intervallo $[-1, 1]$. Una tale n-upla costituisce una buona scelta per una interpolazione su n punti nel suddetto intervallo, in quanto consente una maggiorazione a priori dell'errore di interpolazione. I nodi di Čebyšëv del polinomio n-esimo sono della forma:

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), con\ i = 1, \dots, n$$

Nel file MATLAB *cebishev.m*, grafichiamo la curva data dal polinomio interpolante di quindicesimo grado, utilizzando i nodi di Čebyšëv, e l'andamento dell'errore di interpolazione:

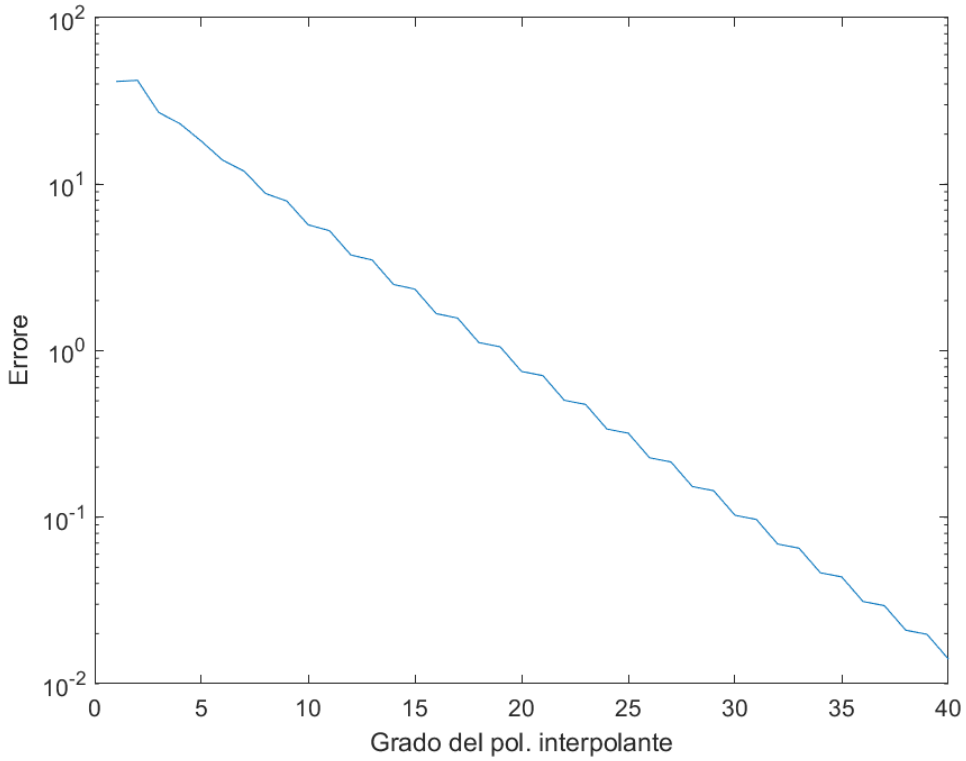
```
runge = @(x) 1./(1+25*x.^2);
x = linspace(-1,1,1e4);
y = runge(x);
n_punti = 16;
cebi = @(i,n) cos(((2*i-1)*pi)./(2*n));
```

```
xp = cebi(1:n_punti, n_punti);
yp = runge(xp);
[v,t] = interp_pol(xp,yp,n_punti-1);
figure(1)
plot(x,y,'b',t,v,'r')
hold on
scatter(xp,yp,'r','filled')
xlabel('x');
ylabel('f(x)')
legend('fun. di Runge','pol. grado 15 (nodi di Čebyšëv)')
hold off
```




```
err = zeros(40,1);
for i = 1:40
    xp = cebi(1:i+1,i+1);
    yp = runge(xp);
    [v,t] = interp_pol(xp,yp,i);
    y = runge(t);
    err(i) = norm(y-v);
end
```

```
figure(2)
semilogy(err);
xlabel('Grado del pol. interpolante')
ylabel('Errore')
```



Un'altra alternativa ai nodi di **Čebyšëv** per migliorare l'errore di interpolazione è l'interpolazione a tratti, ovvero suddividendo l'intervallo di interpolazione in più parti e calcolando su ciascun sotto-intervallo un polinomio interpolante di grado non elevato (ad esempio grado 1 o 2).

CAPITOLO 4

4 Interpolazione polinomiale a tratti

4.1 Interpolazione lineare a tratti

Per evitare i problemi d'interpolazione visti precedentemente, quali:

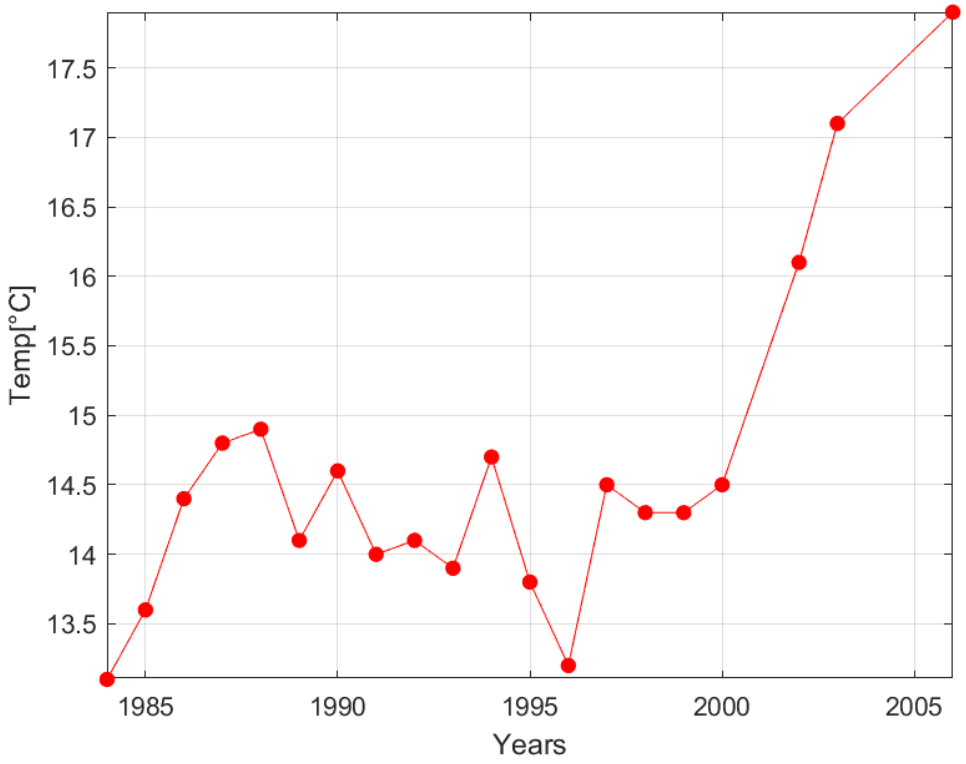
- Mancanza di controllo locale sulla curva interpolante;
- Forti oscillazioni della curva intorno agli estremi del suo dominio;

dovuti entrambi all'utilizzo di polinomi a supporto globale come funzioni interpolanti (soprattutto se di grado elevato), si utilizza un approccio differente. Si introduce un nuovo tipo di modello interpolante/approssimante detto a supporto locale, o a tratti. La peculiarità di tale modello consiste nel partizionare l'insieme su cui sono definiti i dati da rappresentare in k sottoinsiemi, e su ognuno di essi si costruisce la relativa funzione interpolante. Nel caso di **interpolazione polinomiale a tratti** si scelgono come funzioni interpolanti dei polinomi, solitamente di grado basso per evitare i fenomeni di oscillazione. In tal caso, l'intervallo di rappresentazione $[x_1, x_n]$ viene suddiviso in k sotto-intervalli, ognuno dei quali contenente m ascisse relative ad m punti di controllo, e su di ognuno di questi sotto-intervalli viene costruito un polinomio interpolante di grado $m-1$, per un totale di k polinomi. Il caso più semplice di interpolazione polinomiale a tratti consiste nell'utilizzare unicamente polinomi di primo grado, ed è detta **interpolazione lineare a tratti**. In pratica, tale modello prevede di congiungere i punti di controllo a due a due con i segmenti che hanno tali punti come estremi.

Vediamo un esempio di tale tipo di interpolazione nello script file *interp_lineare_a_tratti.m*, utilizzando la function MATLAB *interp1*:

```
load tavg_milan_v2
t = linspace(min(years),max(years),1e4)';
f = interp1(years,temp,t);
plot(t,f,'r')
hold on
scatter(years,temp,'or','filled')
xlabel('Years')
ylabel('Temp[°C]')
```

```
grid on
axis tight
```



CAPITOLO 5

5 Curve B-Spline

Le curve Spline sono delle curve composte da altre curve, e sono per questo dette curve composite. Una curva Spline è costruita congiungendo curve polinomiali mantenendone continuità e regolarità. In genere il tipo di continuità adottato è di tipo C^2, questo significa che l'ultimo punto della curva antecedente ed il primo punto della curva successiva coincidono e la tangente (derivata prima) e la curvatura (derivata seconda) delle due curve in quel punto è identica. In pratica questo significa che in quel punto, in cui le due curve coincidono, si ha lo stesso valore della derivata prima e della derivata seconda. Le B-Splines sono Splines linearmente indipendenti.

L'equazione delle curve B-Spline è la seguente:

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

dove i P_i sono gli $n+1$ punti di controllo e $B_{i,h}(t)$ è l'i-esima funzione B-Spline di grado h definita su un intervallo di nodi $[t_i\ ,\ t_{i+h+1}]$, dove la funzione ha supporto compatto (ovvero il sottoinsieme dei punti del dominio in cui la funzione assume valori non nulli). Dunque, con le curve B-Spline entrano in gioco i nodi, che svolgono un ruolo significativo e definiscono il cosiddetto vettore dei nodi in questo modo:

$$T = (t_0\ , t_1\ , \dots , t_{n+h+1}\ ),\ \ t_i < t_{i+1}$$

Il numero dei nodi allora è così ottenuto:

$$num\_nodi\ =\ num\_punti\ +\ grado\ +\ 1$$

dove

$$grado\ =\ h$$

$$num\_punti = \ n+1\ \ (numero\ dei\ punti\ di\ controllo)$$

Sono state fatte diverse premesse per le curve di Bézier riguardante il grado dei polinomi interpolanti, osserviamo questa diversificazione con un esempio fissando undici punti di controllo: nel caso delle curve B-Spline, il grado dei polinomi interpolanti viene scelto a priori (fissandolo ad esempio a tre ed avendo così quindici nodi); invece nel caso delle curve di Bézier

avremo dei polinomi interpolanti di grado dieci $(n-1)$. Quindi sapendo che l'errore d'interpolazione cresce, a causa dell'instabilità dei polinomi al crescere del grado, si può dire che le curve B-Spline siano in generale un modello più affidabile rispetto alle curve di Bézier.

Analizziamo adesso nel dettaglio chi sono i termini $B_{i,h}(t)$ relativi ad una generica curva B-Spline. Essi sono dei polinomi a tratti, di grado h, calcolati attraverso le formule ricorsive di De Boor:

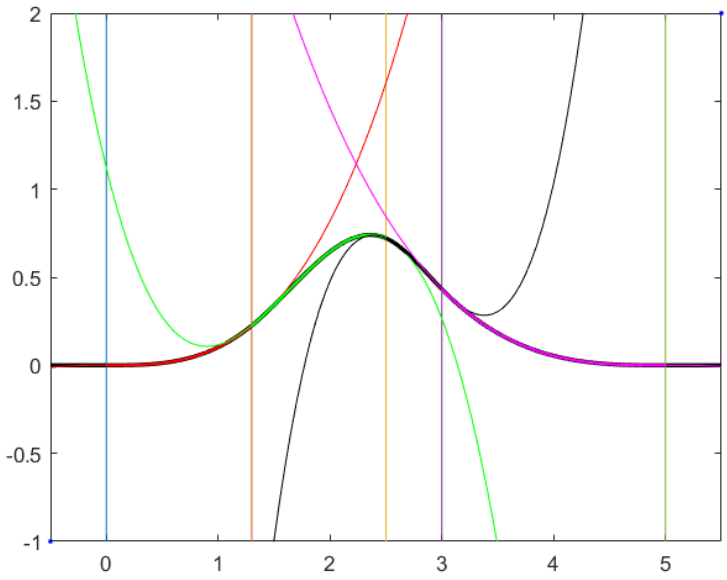
$$con\ h = 1 \quad B_{i,h}(t) = \begin{cases} 1 & se\ t_i \le t < t_{i+1} \\ 0 & altrimenti \end{cases}$$

$$con\ h > 1 \quad B_{i,h}(t) = \frac{t - t_i}{t_{i+h-1} - t_i} B_{i,h-1}(t) + \frac{t_{i+h} - t}{t_{i+h} - t_{i+1}} B_{i+1,h-1}(t)$$

Nel file *script_bspline.m* è presente un esempio di applicazione in MATLAB per plottare una curva B-Spline e le sue funzioni di base, utilizzando la seguente funzione:

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

Di seguito il grafico della curva B-Spline relativo all'esempio:

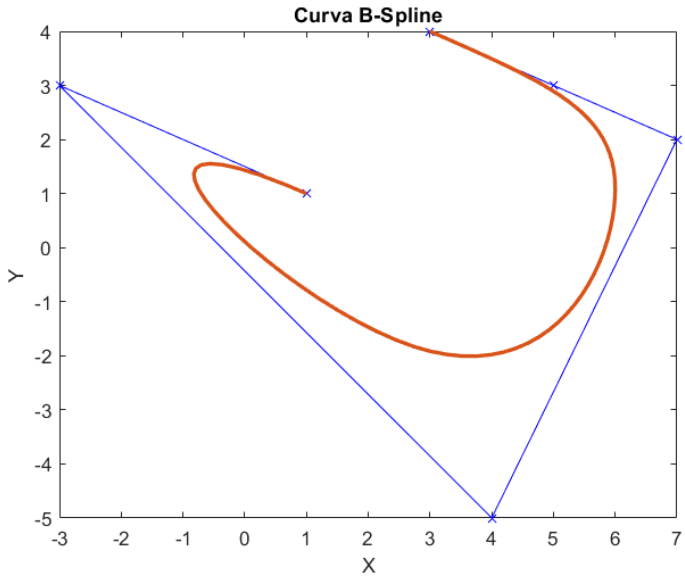


Inoltre MATLAB mette a disposizione un'altra funzione, per rappresentare una curva B-Spline cubica, a partire però da dei fissati punti di controllo:

```
[q, qd, qdd, pp] = bsplinepolytraj(controlPoints,tInterval,tSamples)
```

(per poter utilizzare questa funzione bisogna aver installato su MATLAB il toolbox Robotics System)

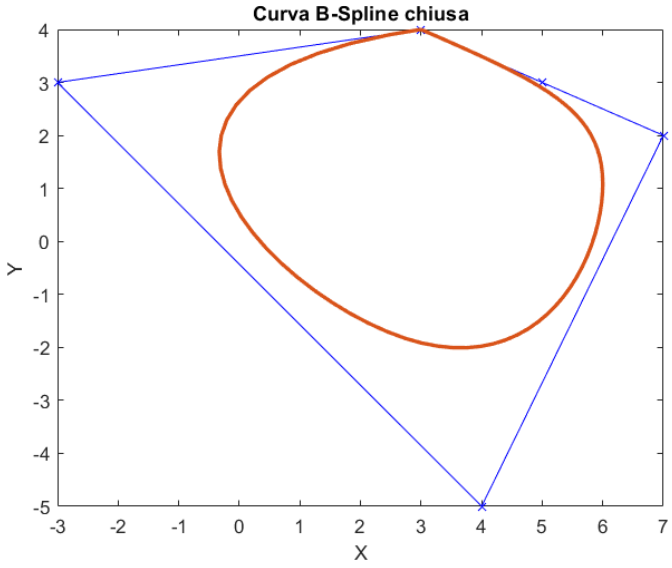
All'interno del file *plot_bspline.m* è presente un esempio per la generazione di una curva B-Spline cubica partendo da dei fissati punti di controllo (in questo esempio i punti di controllo sono `(3,4),(5,2),(7,2),(4,-5),(-3,3) e (1,1))`
.



5.1 Proprietà delle curve B-Spline

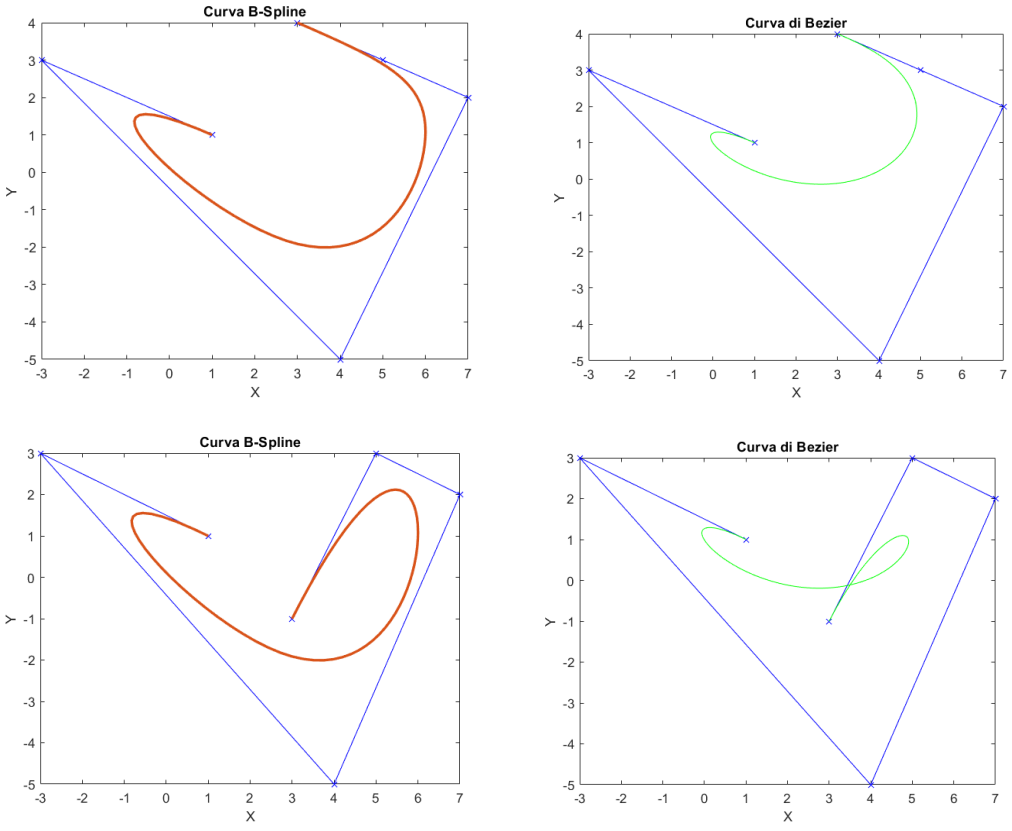
In questo paragrafo verranno elencate una serie di proprietà relative alle curve B-Spline:

- una curva B-Spline può essere chiusa se l'ultimo punto di controllo P_n coincide con il primo P_0 (vedi file *curvachiusa_bspline.m*);



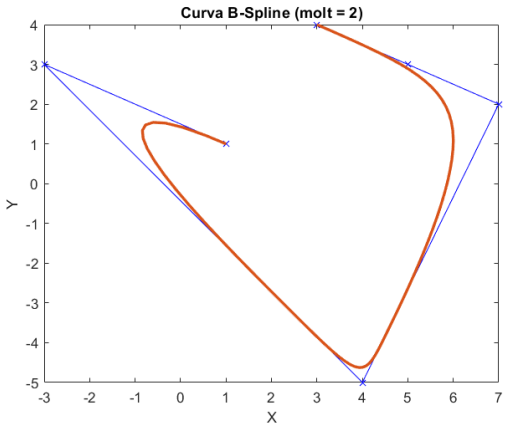
- Proprietà di invarianza: possiamo traslare, ruotare e deformare la curva applicando, ad esempio, delle trasformazioni geometriche o affini solo ai punti di controllo e poi riapplicare la formula per rivalutare la curva (evitandoci di dover effettuare la trasformazione per ogni singolo valore della curva);

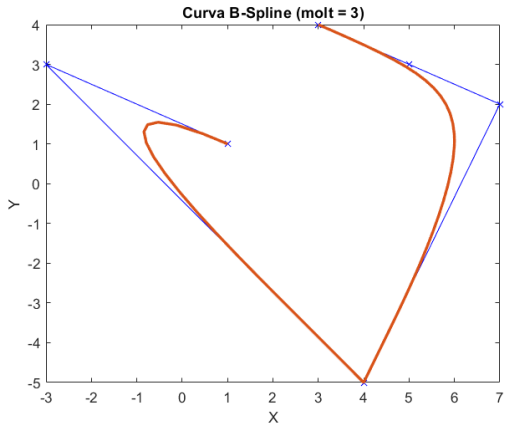
- Controllo locale: come conseguenza del supporto compatto delle B-Spline, se P_j cambia, $C(t)$ risulta modificata solo in corrispondenza dei punti di definizione della B-Spline $B_{j,h}$(vedi file *BézierVSspline)*;



- è possibile modificare la curva in modo tale che sia tangente al primo e all'ultimo segmento nel primo e nell'ultimo punto di controllo, rispettivamente, imponendo che il primo e l'ultimo nodo abbiano molteplicità $h + 1$;
- le curve B-Spline in generale non interpolano alcun punto di controllo;
- fissato il grado h, all'aumentare della molteplicità $(molt)$ di un nodo, la curva si avvicina al nodo multiplo, fino ad interpolare il punto di controllo corrispondente $(molt = h + 1)$, questo perché più si aumenta la molteplicità di un nodo t_i tanto più si riduce il supporto

della B-Spline definita su quel nodo e quindi il risultato è che tende più rapidamente al suo valore massimo (vedi file *bspline_molt.m*);



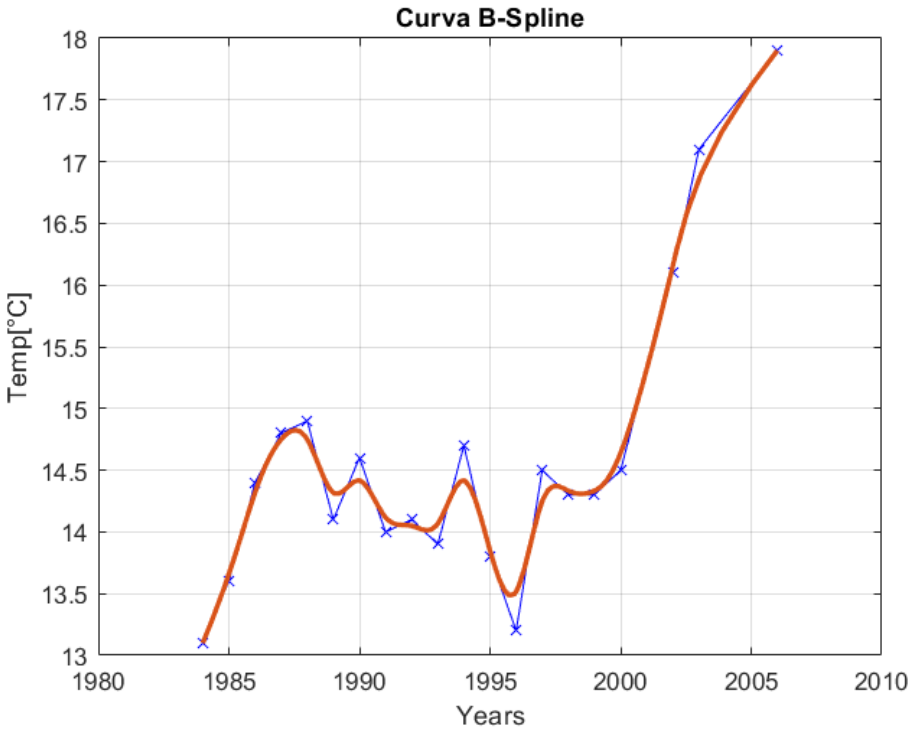


5.2 Applicazione delle B-Spline al caso studio

Passeremo adesso a rappresentare la curva B-Spline cubica del caso studio in esame (vedi file *bspline_casostudio.m*). Nel codice è stata utilizzata la funzione *bsplinepolytraj* che, come già detto in precedenza, permette, a partire da dei fissati punti di controllo, di generare una curva B-Spline cubica relativa a tali punti.

```
%Selezioniamo i punti di controllo
cpts = tavg';
%Scelgo l'intervallo temporale selezionando l’istante iniziale e
%finale
tpts = [0 21];
%Campioni temporali per la traiettoria, specificati come vettore
tvec = linspace(0,21,1e4);

[q,~,~,pp] = bsplinepolytraj(cpts,tpts,tvec);
plot(cpts(1,:),cpts(2,:),'xb-');
title('Curva B-Spline')
xlabel('Years')
ylabel('Temp[°C]')
grid on
hold on
fnplt(pp)
```

Infine, di seguito è riportata la relativa curva B-Spline cubica:



CAPITOLO 6

6 Le NURBS

6.1 I polinomi razionali

Le curve B-Spline, nonostante permettano di rappresentare in modo sufficientemente accurato un'ampia gamma di curve, presentano ancora alcune limitazioni:

- Non è possibile rappresentare esattamente le curve associate a funzioni trigonometriche;
- Non è possibile rappresentare esattamente tutte le sezioni coniche;

Ciò accade perché le funzioni trigonometriche possono essere descritte perfettamente solo da una somma infinita di polinomi (sviluppo in serie di Taylor), mentre le B-Spline sono costituite da una somma finita di polinomi. Esiste però una classe di funzioni polinomiali in grado di descrivere esattamente questi tipi di curve: i **polinomi razionali**.

Si decise dunque, per ampliare la varietà di curve rappresentabili, di estendere le B-Spline tramite l'impiego di polinomi razionali, ottenendo così, nel 1975 circa, presso la BOEING Corporation, le curve **NURBS (Non Uniform Rational B-Splines)**. L'aggettivo "Non Uniform" si riferisce alla possibilità che il vettore dei nodi della curva sia non uniforme, mentre l'aggettivo "Rational" fa riferimento all'impiego di polinomi razionali.

6.2 Formulazione matematica

Una curva NURBS è definita da quattro caratteristiche: il **grado**, i **vertici di controllo**, il **vettore dei nodi** e il **vettore dei pesi**.

Il grado è un numero intero positivo h, solitamente variabile tra 1 e 5, per evitare il fenomeno delle oscillazioni, anche se teoricamente esso può assumere un qualunque valore intero positivo. A volte, si può far riferimento all'**ordine** di una curva NURBS, ossia un numero intero positivo pari al grado della curva più uno.

$$ordine = h + 1$$

I vertici di controllo sono una serie di punti $P_0, \dots, P_n$ in numero almeno pari a $h + 1$, che hanno la funzione di fare da guida alla curva da rappresentare. Ad ogni vertice è associato un peso

$w_i \geq 0$ $(con\ i = 0, \dots, n)$ che indica la capacità di tale vertice di attrarre a sé la curva. Quando i vertici di controllo di una curva hanno tutti lo stesso peso (solitamente 1), la curva viene denominata "non razionale"; in caso contrario, è detta razionale.

Il vettore dei nodi è un vettore numerico $T = (t_0, t_1, \dots, t_m)$ costituito da $m + 1$ elementi. In particolare, il valore di m deve essere pari ad:

$$m = n + h + 1$$

dove $n + 1$ è il numero di vertici di controllo, ed h è il grado della curva.

Il vettore dei nodi deve soddisfare diverse condizioni tecniche. Di norma, per assicurarsi che queste condizioni tecniche siano soddisfatte, si richiede che i valori numerici del vettore siano ordinati in ordine crescente e che ogni nodo abbia molteplicità al massimo pari al grado h della curva. 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 al grado della curva più uno $h + 1$. Tuttavia, al crescere della molteplicità di un nodo la curva perde di regolarità su di un dato intervallo, fino a diventare discontinua quando la molteplicità di un nodo è esattamente pari ad $h + 1$. Si dice che un nodo ha **molteplicità piena** se il suo valore si ripete tante volte quante il grado h della curva, mentre un nodo si dice **semplice** se esso ha molteplicità pari a 1. Se una sequenza di nodi inizia con un nodo a molteplicità piena, segue con dei nodi semplici, termina con un nodo a molteplicità piena e tutti i valori sono ugualmente spaziati, i suoi nodi sono detti **uniformi**. Viceversa, il vettore dei nodi è detto **non uniforme**. Se $t_0 = 0\ e\ t_m = 1$ il vettore dei nodi T è detto "standard".

Una volta presentati tutti i parametri che caratterizzano una curva NURBS, si riporta la sua formulazione matematica: dati i punti di controllo $P_0, \dots, P_n$, con i relativi pesi $w_0, \dots, w_n$, ed il vettore di nodi $T = (t_0, \dots, t_m)$, con $m = n + h + 1$, la curva NURBS parametrica $C(t)$ sarà pari a:

$$C(t) = \frac{\sum_{i=0}^{n} w_i * P_i * B_{i,h}(t)}{\sum_{i=0}^{n} w_i * B_{i,h}(t)}\ , con\ t \in [t_0, t_m]$$

dove con $B_{i,h}(t)$ si indicano le funzioni di base B-Spline di grado h.

Si può utilizzare anche una forma equivalente quale:

$$C(t) = \sum_{i=0}^{n} P_i * R_{i,h}(t)\ , con\ t \in [t_0, t_{n+h+1}]$$

dove:

$$R_{i,h}(t) = \frac{w_i * B_{i,h}(t)}{\sum_{i=0}^{n} w_i * B_{i,h}(t)} \text{ , con } t \in [t_0, t_{n+h+1}]$$

6.3 Proprietà

Le curve NURBS possiedono le seguenti proprietà:

- La funzione $R_{i,h}(t)$ è una funzione razionale di grado h in t;
- La funzione $R_{i,h}(t)$ è non negativa $\forall\ i, h, t$;
- Supporto locale: la funzione $R_{i,h}(t)$ assume valori non nulli solo nell'intervallo $[t_i, t_{i+h+1}]$;
- La somma di tutte le funzioni di base $B_{i,h}(t)$ su ogni intervallo $[t_i, t_{i+h+1}]$ è pari a 1;
- In presenza di nodi multipli, con molteplicità pari a k, la curva di NURBS $C(t)$ è una funzione di classe C^{h-k};
- Dato un nodo di molteplicità $h + 1$, la curva NURBS passa esattamente per il punto di controllo associato a tale nodo;
- Sia P_i un punto di controllo di molteplicità pari a k, allora la curva NURBS $C(t)$ risultante sarà una funzione di classe C^{h-k};
- Dato un punto di controllo P_i di molteplicità $k = h + 1$, allora la curva $C(t)$ passerà esattamente per il punto P_i;
- Strong convex-hull: la curva $C(t)$ è contenuta nel dominio convesso generato dai suoi punti di controllo (se $w_i \geq 0\ \forall i$);

Una proprietà molto importante che possiedono le curve NURBS è l'**invarianza alle trasformazioni affini.** Una trasformazione affine non è altro che una trasformazione lineare seguita da una traslazione. Dato un generico punto di controllo P, indichiamo una trasformazione affine per tale punto come: $A[P] = L[P] + T$.

Se indichiamo una generica curva NURBS con $C(t)$, allora:

$$C(t) = \sum_i P_i * R_{i,h}(t)$$

$$A[C(t)] = L[C(t)] + T = \sum_i \{L[P_i] * R_{i,h}(t)\} + T$$

Se invece consideriamo la curva NURBS costruita con le trasformate affini dei punti di controllo:

$$C'(t) = \sum_i A[P_i] * R_{i,h}(t) = \sum_i \{L[P_i] * R_{i,h}(t)\} + T \sum_i R_{i,h}(t)$$

$$\sum_i R_{i,h}(t) = 1$$

$$C'(t) = \sum_i \{L[P_i] * R_{i,h}(t)\} + T = A[C(t)]$$

Dunque:

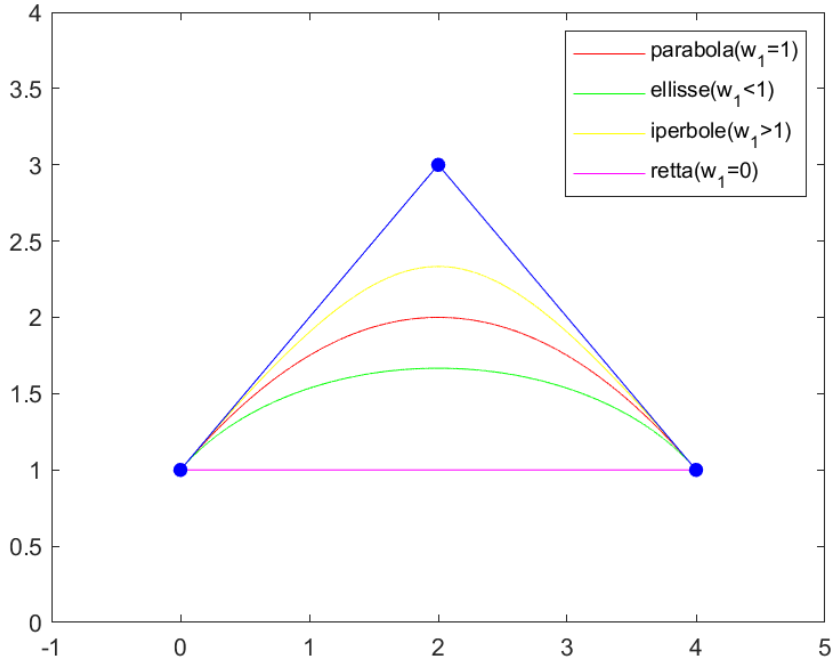
$$A[C(t)] = \sum_i A[P_i] * R_{i,h}(t)$$

cioè l'immagine affine di una curva NURBS si ottiene trasformando i punti di controllo e lasciando i pesi inalterati.

Come ultima proprietà, mostriamo come una qualunque sezione conica può essere rappresentata mediante una curva NURBS di grado 2, con vettore dei nodi $T = (0,0,0,1,1,1)$ e tre punti di controllo P_0, P_1, P_2 , lavorando unicamente sul peso w_1(avendo posto $w_0 = w_2 = 1$). In particolare, se:

- $w_1 = 1$, otteniamo una parabola;
- $w_1 < 1$, otteniamo un'ellisse;
- $w_1 > 1$, otteniamo una iperbole;
- $w_1 = 0$, otteniamo una retta.

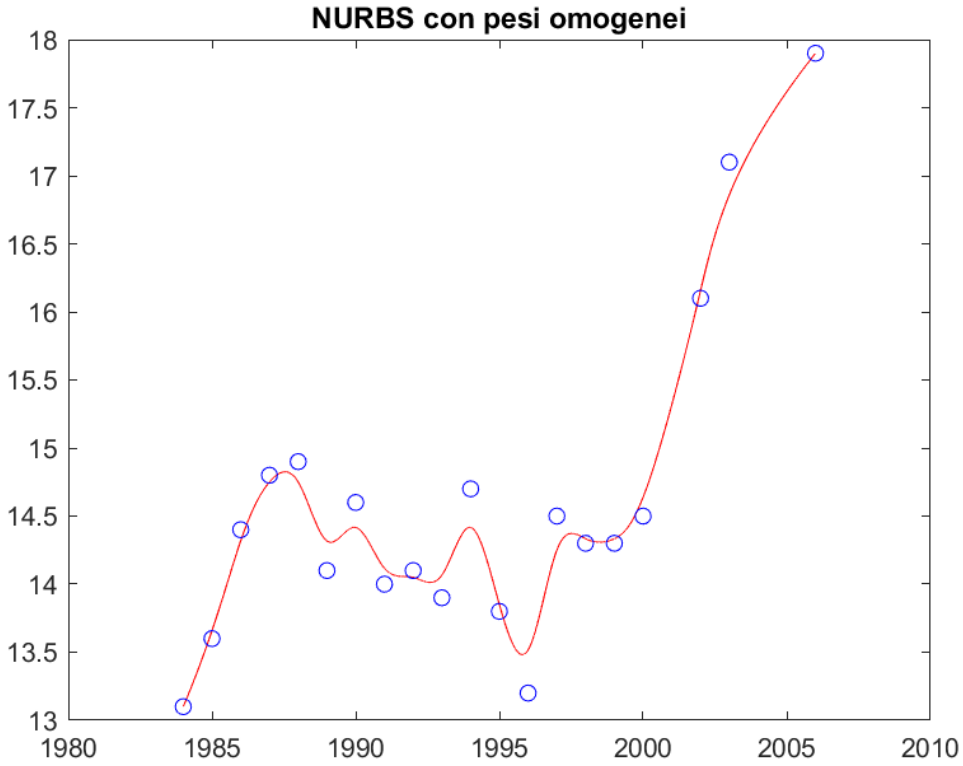
Come esempio si veda lo script file *nurbs_w.m* .

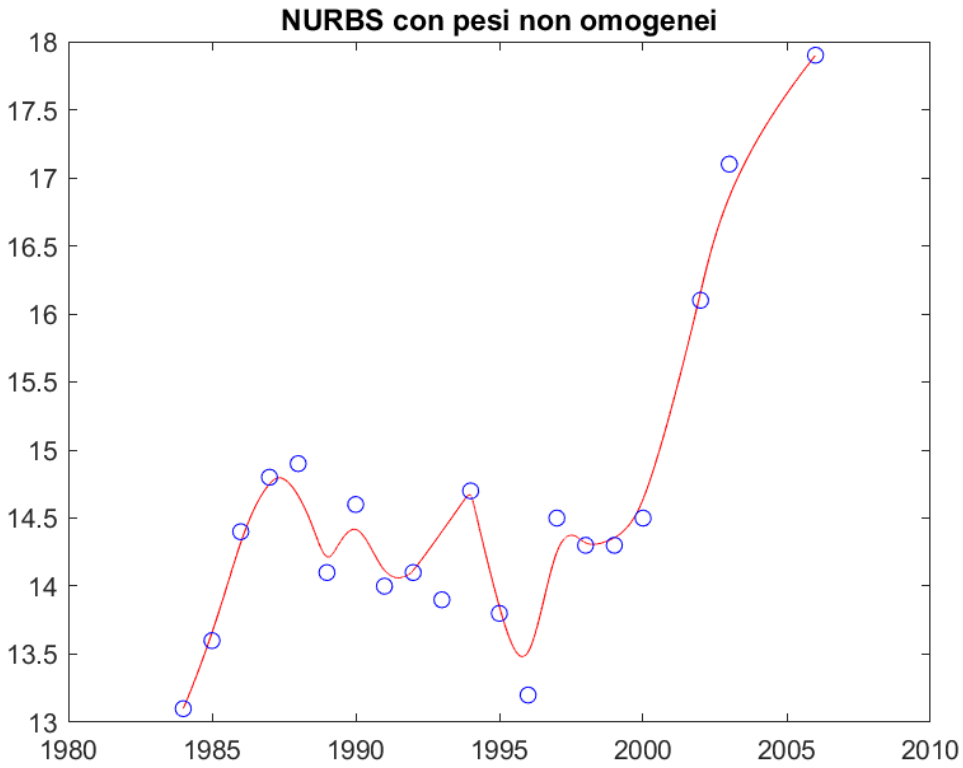


6.4 Applicazione delle NURBS sul caso studio

Rappresentiamo i dati del nostro caso studio sulla temperatura media annuale della città di Milano, utilizzando come modello approssimante una curva NURBS di quarto ordine (terzo grado), come riportato nello script file *nurbs_cs.m* . In particolare, viene effettuata una prima rappresentazione dei dati, attribuendo a tutti i punti lo stesso peso w_i (pari a 1), ed una seconda in cui si ritengono alcuni valori di temperatura più affidabili rispetto ad altri, e ciò si traduce nel modello matematico in un vettore dei pesi w non omogeneo, che assume valori numerici più elevati in corrispondenza dei dati ritenuti più affidabili, e viceversa per i dati meno affidabili.

```
load tavg_milan_v2
p = tavg';
n_punti = length(p(1,:));
h = 3;
w1 = ones(1,n_punti);
t = [zeros(1,h+1) 1/(n_punti-h)*(1:n_punti-h-1) ones(1,h+1)];
u = linspace(0,1,1e4);
c = nurbsfun(h+1,t,w1,p,u);
plot(c(1,:),c(2,:),'r');
title('NURBS con pesi omogenei')
hold on
scatter(p(1,:),p(2,:),'bo')

%utilizzando pesi disomogenei
w2 = [0.2*ones(1,5) ones(1,4) 0 5*ones(1,5) ones(1,5)];
c = nurbsfun(h+1,t,w2,p,u);
figure(2)
plot(c(1,:),c(2,:),'r');
title('NURBS con pesi non omogenei')
hold on
scatter(p(1,:),p(2,:),'bo')
```


NURBS con pesi omogenei
18
17.5
17
16.5
16
15.5
15
14.5
14
13.5
13
1980
1985
1990
1995
2000
2005
2010


NURBS con pesi non omogenei
18
17.5
17
16.5
16
15.5
15
14.5
14
13.5
13
1980
1985
1990
1995
2000
2005
2010

CAPITOLO 7

7 Interpolazione e qualità delle immagini

In questo capitolo vedremo come è possibile, attraverso l'interpolazione, migliorare la qualità di immagini sgranate. Per effettuare tale analisi, carichiamo un'immagine e ridimensioniamola in modo tale da ottenere un'immagine di scarsa qualità (con pochi pixel):

```
%Caricamento immagine ridimensionata
img = imresize(imread('cane.jpg'), [64 NaN]);
[r,c,v] = size(img);
```

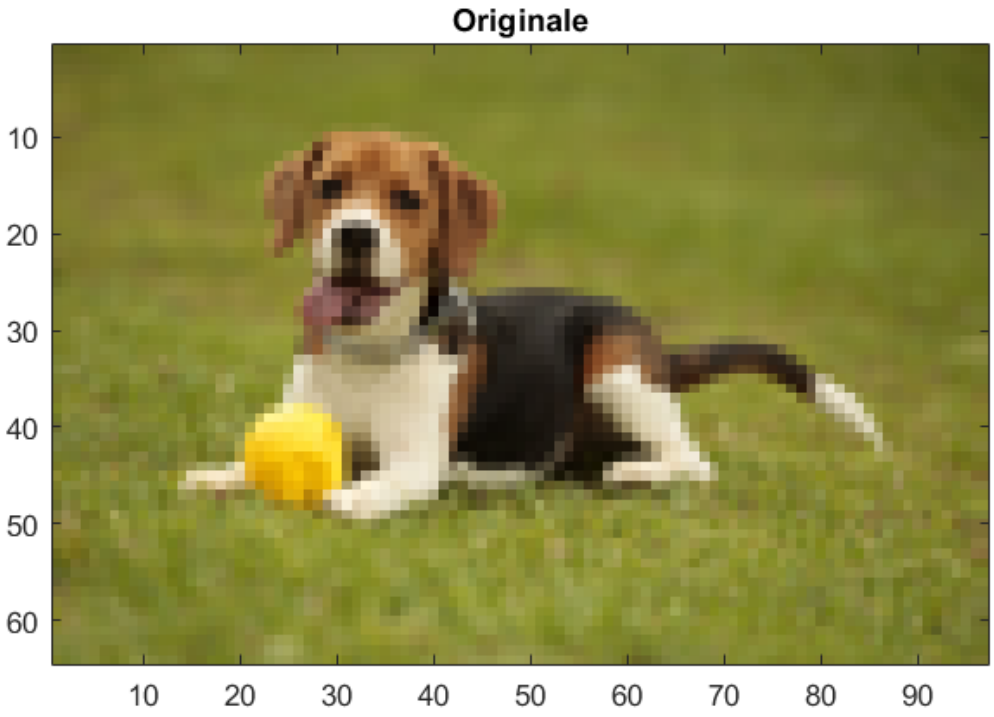
Preallochiamo le nuove immagini e generiamo una griglia di pixel più fitta. Saranno preallocate due immagini, poiché mostreremo un miglioramento dell'immagine sia attraverso l'interpolazione lineare sia attraverso l'interpolazione Spline:

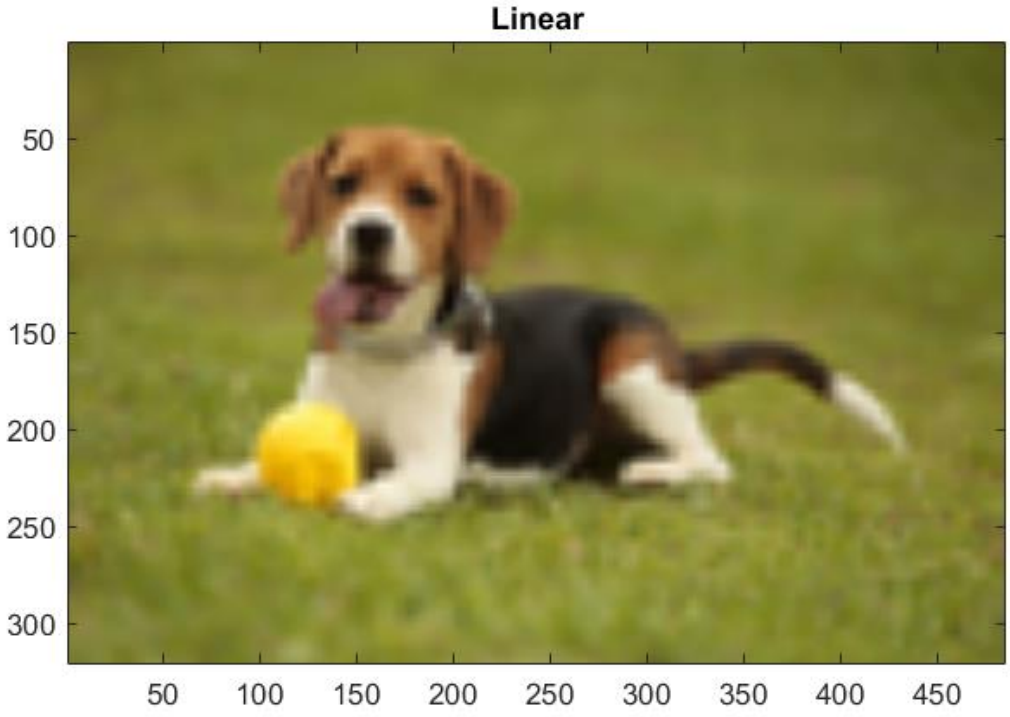
```
%Preallocazione delle nuove immagini
NEW_I = zeros(5*r,5*c,v,'uint8');
NEW_I_L = zeros(5*r,5*c,v,'uint8');

%Determino la griglia dell'immagine   di  partenza
x = 1:c;
y = 1:r;

%Determino la griglia dell'immagine  che voglio  ottenere
new_x = linspace(1,c,5*c)';
new_y = linspace(1,r,5*r);
```

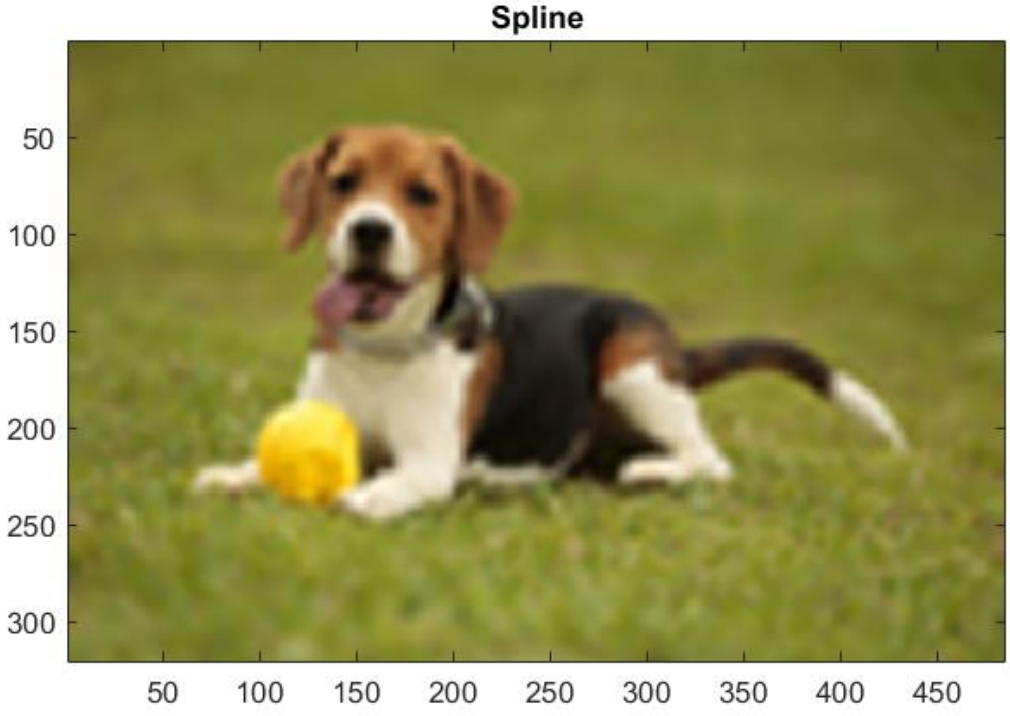
A questo punto, possiamo effettuare l'interpolazione sulla griglia originale, attraverso la funzione *interp2:*

```
%Generazione tramite interpolazione delle due nuove immagini
for i=1:3

NEW_I(:,:,i)=uint8(interp2(x,y,double(img(:,:,i)),new_x,new_y,'Spl
ine'));

NEW_I_L(:,:,i)=uint8(interp2(x,y,double(img(:,:,i)),new_x,new_y,'l
inear'));
end
```

Infine, rappresenteremo le immagini ottenute a seguito delle diverse interpolazioni, evidenziando i miglioramenti apportati:




Linear
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450


Spline
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450