Lezioni di Calcolo Numerico
Lezione 05: Approssimazione di dati e funzioni

Alberto Tibaldi

5 maggio 2018

Indice

3 Interpolazione polinomiale a tratti: spline **1**
3.1 Introduzione alle spline 1
3.2 Spline di ordine 1 2
3.3 Spline cubiche 3
3.4 Cenni al metodo dei trapezi 6

A Esempio di uso di spline lineari **7**

B Esempio di uso di spline cubiche **8**

C Approfondimenti sulle spline **8**
C.1 Quando abbiamo pochi nodi: spline, polinomi interpolanti, o *entrambi*? 8
C.2 Ma come fa MATLAB a calcolare le spline?! 9

3 Interpolazione polinomiale a tratti: spline

3.1 Introduzione alle spline

Fino a questo momento studiando questo argomento ci siamo concentrati sull'interpolazione polinomiale, ossia sull'idea di approssimare una funzione f definita su un intervallo $[a, b]$ mediante un **unico** polinomio di grado n. A questo fine, abbiamo partizionato il nostro intervallo in $n+1$ punti e, mediante procedure analitiche (rappresentazione di Lagrange) o numeriche (il comando `polyfit`) abbiamo ricavato i coefficienti del polinomio.

Tuttavia, esiste un secondo principio che possiamo applicare al fine di approssimare una funzione. Immaginiamo, proprio come prima, di aver partizionato l'intervallo $[a, b]$ in n sotto-intervalli[1]. Al posto di utilizzare un unico polinomio definito su tutto l'intervallo, la scelta che discuteremo in questa sezione sarà **utilizzare un polinomio diverso e di grado basso su ciascun sotto-intervallo**. Questo significa, in altre parole, utilizzare come approssimante una **funzione polinomiale a tratti**: la funzione di partenza è approssimata da un polinomio di grado d **caratterizzato da coefficienti tra loro diversi su ciascun intervallo**; il grado massimo d del polinomio è lo stesso in tutti i sotto-intervalli. A questo punto, lo studente impaziente, leggendo questa definizione, potrebbe pensare

> «C'è qualcosa che mi puzza: la grande idea del giorno è usare tanti polinomi diversi su ciascun sotto-intervallo e che non c'entrano nulla l'uno con l'altro. Bah. Almeno prima avevamo un unico polinomio che almeno sappiamo essere liscio su tutto l'intervallo $[a, b]$. Cioè, ma chi ci dice che ora sui punti di raccordo tra due polinomi, questi non facciano qualcosa di pazzo?!»

[1] dire di partizionare un intervallo in n sotto-intervalli è esattamente coincidente a dire che lo si partiziona in $n+1$ nodi; se per esempio volessimo $n+1=5$ nodi, allora l'intervallo sarebbe partizionato in 4 sotto-intervalli, in 4 segmenti

In effetti, l'idea che vorremmo presentare è ancora incompleta. Con riferimento ai dubbi del nostro studente impaziente, è sì vero che vogliamo usare su ciascun sotto-intervallo polinomi diversi, con coefficienti diversi, ma non ricordo di aver scritto che **debbano essere indipendenti l'uno dall'altro**. Ciò che in effetti distingue un banale insieme di funzioni polinomiali a tratti dalle funzioni che intendiamo definire è proprio **l'imposizione di condizioni di raccordo tra coppie di polinomi adiacenti**. Queste condizioni di raccordo fanno dunque sì che i polinomi siano diversi ma che, d'altra parte, ciascun polinomio *cominci e termini avendo la consapevolezza* di quello che sta per fare il precedente/successivo. Introdurre questo rapporto tra polinomi adiacenti permette alle funzioni che useremo, dette **spline**, da un lato una certa *libertà di movimento*, poiché ciascun sotto-intervallo ha un polinomio diverso, ma anche una certa *armonia*, poiché ciascuna funzione *sa cosa sta facendo* la funzione successiva.

Il nostro compito adesso è cercare di tradurre in un linguaggio più formale l'idea appena espressa. Volendo una definizione intermedia tra il discorso appena fatto e una formulazione rigorosa, si definisce una spline di ordine d interpolante f nei nodi $\{x_i\}$ una funzione $S_d(x)$ polinomiale a tratti con proprietà di regolarità nei punti di raccordo. Più nel dettaglio, $S_d(x)$ deve soddisfare le condizioni ora riportate.

1. $S_d(x)$ è un polinomio di grado (al più) pari a d su ciascun sotto-intervallo $[x_i, x_i + 1]$, per $i = 1, ..., n$.

2. La spline deve essere interpolante, quindi soddisfare le condizioni di interpolazione

$$S_d(x_i) = y_i, \qquad i = 1, ..., n + 1.$$

3. La derivata k-esima $S_d^{(k)}(x)$, per $k = 0, 1, ..., d - 1$, è una funzione continua nell'intero intervallo $[a, b]$.

3.2 Spline di ordine 1

Prima di passare al piatto forte, è opportuno introdurre quello che sicuramente è l'esempio *più semplice* di spline: la spline lineare, ovvero, $d = 1$. Si noti che questa è esattamente la stessa cosa che fa il comando `plot` ogni volta che utilizziamo un numero di nodi molto basso per disegnare una funzione: si ottiene un grafico *squadrato*, che in effetti altri non è che la spline lineare *su pochi punti* della funzione. In altre parole, l'approssimazione consiste nel nel rappresentare la funzione di partenza unendo mediante segmenti di retta i vari nodi di interpolazione.

La spline lineare $S_1(x)$ soddisfa le condizioni presentate al termine della Sezione 3.1: $S_1(x)$ è continua, ma non derivabile, dal momento che $d = 1$ e tutte le derivate fino alla $(d - 1)$-esima sono continue (quindi, fino alla derivata k-esima con $k = 0$, che significa continuità e nient'altro). D'altra parte se approssimiamo la funzione su ciascun intervallo mediante una retta, allora da un intervallo a un altro le rette dovranno cambiare pendenza, altrimenti non sarà possibile che seguano il profilo della funzione! Ma quindi, la derivata sarà certamente -e giustamente- discontinua!

L'espressione matematica della spline lineare è molto semplice, e si può ricavare mediante la solita dimostrazione che ho imparato in terza liceo su come trovare la retta passante per due punti: proprio di questo si parla! In particolare, si ha che

$$\frac{S_1(x) - f(x_i)}{f(x_{i+1}) - f(x_i)} = \frac{x - x_i}{x_{i+1} - x_i}, \qquad x \in [x_i, x_{i+1}]$$

che, dopo alcune semplicissime manipolazioni algebriche, permette di ottenere

$$S_1(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i), \qquad x \in [x_i, x_{i+1}]. \tag{1}$$

In ogni caso, non è strettamente necessario, su MATLAB®, implementare questa funzione. In effetti, esiste il comando `interp1` che permette esattamente di ottenere il risultato garantito da (1). Per concludere, si segnala che l'Appendice A fornisce un codice MATLAB®in grado di generare, per diversi nodi di interpolazione, la spline lineare interpolante.

Studiando il precedente argomento (interpolazione polinomiale), abbiamo notato che possono esserci problemi di convergenza. Per le spline lineari, ci viene incontro un teorema, che ora

enunceremo. Data una partizione $a = x_1 < x_2 < ... < x_{n+1} = b$, sia S_1 la spline interpolante una funzione $f \in C^{(2)}([a,b])$ nei nodi della partizione x_i, $i = 1, ..., n+1$. Dato

$$h = \max_{1 \le i \le n} (x_{i+1} - x_i),$$

allora

$$\|f - S_1\|_\infty = \mathcal{O}(h^2), \quad h \to 0.$$

Detto in altre parole, data una qualsiasi funzione a derivata seconda continua sull'intero intervallo, l'approssimazione mediante una spline lineare converge **sempre** uniformemente al crescere del numero di nodi (o, detto in altra maniera, al **decrescere della massima distanza tra nodi** h), e con un tasso di convergenza pari al quadrato di h.

3.3 Spline cubiche

«Siamo contenti?»

Mah. Abbastanza, diciamo. Sicuramente poteva andarci peggio. Dobbiamo riconoscere che le spline lineari ci danno delle garanzie: aumentando il numero di nodi, possiamo approssimare la nostra funzione arbitrariamente bene, ossia bene quanto ci piace. Alla fine, questo è un po' lo stesso principio che usiamo al momento di disegnare una funzione con `plot`: se vediamo la funzione troppo *squadrata* rispetto a quello che ci aspetteremmo, possiamo sempre provare ad aumentare il numero di nodi, e il disegno diventerà più liscio.

Se però la domanda fosse stata

«Siamo entusiasti?»

La risposta sarebbe stata no. Abbiamo la garanzia che il tasso di convergenza sia $\mathcal{O}(h^2)$ indipendentemente dalla regolarità della funzione f (o quasi: deve avere almeno la derivata seconda continua su $[a,b]$!). Tuttavia, a noi tutto sommato farebbe piacere che, se la f fosse bella liscia, con insomma molte derivate continue, allora non fosse necessario usare troppi nodi. Avere un h basso significa usare molti nodi, ma ricordiamoci sempre che calcolare queste spline ha comunque un certo costo computazionale, ossia, richiede un certo numero di operazioni.

Tuttavia, con $d = 1$ non è possibile fare di meglio. Ma non ce l'ha mica detto il dottore, che le spline lineari siano l'unica possibilità a disposizione. In effetti, quando si parla di spline, uno pensa sempre alle **spline cubiche**, ovvero $d = 3$. Questo significa che, per ogni j-esimo sotto-intervallo, la nostra spline avrà una forma del tipo

$$S_3(x) = a_j x^3 + b_j x^2 + c_j x + d_j, \qquad x \in [x_j, x_{j+1}]. \tag{2}$$

Abbiamo dunque implicitamente definito il j-esimo sotto-intervallo come quello compreso tra x_j e x_{j+1}. Dal momento che ipotizziamo sempre di lavorare con $n+1$ nodi, avremo ancora n intervalli. In altre parole, al fine di determinare i 4 coefficienti a_j, b_j, c_j, d_j per ognuno degli n sotto-intervalli, avremo bisogno di $4n$ condizioni che le nostre spline devono rispettare.

Non è che dobbiamo inventarci noi le condizioni che la spline deve rispettare: ne abbiamo un elenco al termine dell'introduzione in Sezione 3.1! Volendole scrivere per esteso per quanto riguarda il caso in esame, $d = 3$,

1. Come già espresso in (2), $S_3(x)$ è un polinomio cubico su ciascun sotto-intervallo.
2. La spline deve essere interpolante e quindi, per ogni nodo della partizione, deve soddisfare le condizioni di interpolazione

   $$S_3(x_i) = y_i, \qquad i = 1, ..., n+1.$$

   Questa espressione può essere scritta in maniera più esplicita come

   $$a_j x_j^3 + b_j x_j^2 + c_j x_j + d_j = y_j, \qquad j = 1, ..., n,$$

che, tradotto dal matematichese, significa: *fai in modo che l'estremo sinistro di ognuno dei polinomi sia agganciato a* (x_i, y_i). C'è un'ulteriore condizione di interpolazione, che è

$$a_n x_{n+1}^3 + b_n x_{n+1}^2 + c_n x_{n+1} + d_n = y_{n+1}, \tag{3}$$

che va imposta separatamente, dal momento che noi abbiamo scelto di far sì che sia *l'estremo sinistro* di ciascun polinomio a soddisfare la condizione di interpolazione, ma, non essendoci un polinomio *a destra dell'ultimo*, l'ultima condizione va imposta a parte[2]

3. Da un lato, dalla precedente scelta, sembrerebbe che ciascun polinomio funzioni solo *a sinistra*, ma *a destra* sia libero. In effetti, però, lo scopo delle condizioni di raccordo è proprio quello di *dare armonia* e far sì che i polinomi *vadano d'accordo* sull'intero intervallo $[a, b]$. La prima di queste condizioni *di armonia* o, più formalmente, di raccordo, è quella di continuità: la spline deve essere continua su ciascun punto di raccordo; quindi,

$$a_j x_{j+1}^3 + b_j x_{j+1}^2 + c_j x_{j+1} + d_j = a_{j+1} x_{j+1}^3 + b_{j+1} x_{j+1}^2 + c_{j+1} x_{j+1} + d_{j+1}, \qquad j = 1, ..., n-1. \tag{4}$$

Qui scegliamo che *l'estremo destro* di ciascun j-esimo polinomio sia uguale *all'estremo sinistro* del successivo. Come conseguenza dell'identificazione del j-esimo intervallo come definito tra i nodi $[x_j, x_{j+1}]$, questa condizione va soddisfatta solamente per j che va da 1 a $n-1$, dal momento che in questo modo la condizione di continuità sarà imposta solo per i nodi compresi da x_2 a x_n; non avrebbe senso imporre una condizione di continuità su x_1 o x_{n+1}, dal momento che a sinistra di x_1 e a destra di x_{n+1} non abbiamo altri polinomi.

4. Dal momento che $d = 3$, anche la derivata prima deve essere continua; di conseguenza, l'espressione della derivata sarà, per ciascun sotto-intervallo,

$$\frac{\mathrm{d}S_3(x)}{\mathrm{d}x} = 3a_j x^2 + 2b_j x + c_j, \qquad j = 1, ..., n.$$

Esattamente come per la condizione di continuità sarà dunque necessario imporre la condizione di continuità della derivata. Questo porta a un'espressione analoga alla precedente, che sarà

$$3a_j x_{j+1}^2 + 2b_j x_{j+1} + c_j = 3a_{j+1} x_{j+1}^2 + 2b_{j+1} x_{j+1} + c_{j+1}, \qquad j = 1, ..., n-1. \tag{5}$$

5. Di nuovo, essendo $d = 3$, anche la derivata seconda dovrà essere continua sull'intero intervallo $[a, b]$. Quindi, si può scrivere la derivata seconda per ciascun sotto-intervallo come

$$\frac{\mathrm{d}^2 S_3(x)}{\mathrm{d}x^2} = 6a_j x + 2b_j, \qquad j = 1, ..., n.$$

E quindi, proprio come per le condizioni di continuità e derivabilità, dovremo imporre

$$6a_j x_{j+1} + 2b_j = 6a_{j+1} x_{j+1} + 2b_{j+1}, \qquad j = 1, ..., n-1. \tag{6}$$

«Tutto qui?»

Purtroppo, temo di no. Ricordiamoci che abbiamo bisogno di $4n$ condizioni; le abbiamo trovate? Contiamo un po':

$$\underbrace{n+1}_{\text{interp.}} + \underbrace{n-1}_{\text{cont.}} + \underbrace{n-1}_{\text{der.I}} + \underbrace{n-1}_{\text{der.II}} = 4n-2.$$

Al fine di garantire l'unicità della spline (e quindi un metodo che possa permetterci di trovarne una, senza ambiguità), avremmo voluto $4n$ condizioni e invece, seguendo le indicazioni forniteci dalle definizioni di spline, ne abbiamo trovate solo $4n-2$: abbiamo bisogno di altre due condizioni, che dobbiamo decidere noi.

Nella letteratura, vengono proposte, per poter arrivare a $4n$, tre categorie di condizioni aggiuntive, ora presentate.

[2] tutta questa è stata una scelta arbitraria, ma che non cambia in alcun modo il risultato finale: avremmo potuto richiedere che fossero gli estremi destri di ciascun polinomio a soddisfare la condizione di interpolazione, e il primo sarebbe stato trattato separatamente, o chissà quale altra combinazione: non cambia nulla.

1. Spline cubiche naturali: si ottengono imponendo, come due condizioni aggiuntive, la derivata seconda uguale a zero ai nodi estremi:

$$\begin{aligned} 6a_1x_1 + b_1 &= 0 \\ 6a_nx_{n+1} + b_n &= 0. \end{aligned}$$

2. Spline cubiche *not-a-knot*: si ottengono imponendo, come condizioni aggiuntive, la continuità della derivata *terza* nei nodi x_2 e x_n. Questo significa che

$$\frac{\mathrm{d}^3 S_3(x)}{\mathrm{d}x^3} = 6a_j, \qquad j = 1, ..., n.$$

   In altre parole stiamo chiedendo di imporre le condizioni

$$\begin{aligned} a_1 &= a_2 \\ a_{n-1} &= a_n. \end{aligned}$$

   Questa condizione, che è usata di default in MATLAB®, è molto particolare. Infatti, la condizione si chiama *not-a-knot*, ovvero *non un nodo*, perché di fatto, sui primi due e sugli ultimi due intervalli, la spline cubica interpolante diventerà un polinomio interpolante di grado 3. Infatti, se abbiamo due polinomi di grado 3, imporre la condizione **anche** sulla derivata terza significa imporre **l'eguaglianza di tutti i coefficienti**. Questo si può dimostrare semplicemente rivedendo quali sono le condizioni di continuità imposte tra i polinomi appartenenti al primo e al secondo intervallo:

$$\begin{aligned} a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 &= a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 \\ 3a_1x_2^2 + 2b_1x_2 + c_1 &= 3a_2x_2^2 + 2b_2x_2 + c_2 \\ 6a_1x_2 + 2b_1 &= 6a_2x_2 + 2b_2 \\ 6a_1 &= 6a_2. \end{aligned}$$

   Sostituendo *a gambero* l'ultima condizione nella penultima, si ottiene $b_1 = b_2$; sostituendo le ultime due condizioni nell'ultima, si ottiene $c_1 = c_2$; sostituendo queste tre nella prima, si ottiene infine $a_1 = a_2$. Per l'ultimo intervallo vale una dimostrazione analoga.

3. Spline cubiche vincolate: ammesso di conoscere per qualche motivo il valore delle derivate prime $f'(x)$ agli estremi x_1 e x_{n+1}, si richiede che la derivata prima della spline agli estremi sia uguale a quella della funzione, ossia

$$\begin{aligned} \left.\frac{\mathrm{d}S_3(x)}{\mathrm{d}x}\right|_{x=x_1} &= f'(x_1) \\ \left.\frac{\mathrm{d}S_3(x)}{\mathrm{d}x}\right|_{x=x_{n+1}} &= f'(x_{n+1}). \end{aligned}$$

   Queste condizioni si possono scrivere in maniera più esplicita come

$$\begin{aligned} 3a_1x_1^2 + 2b_1x_1 + c_1 &= f'(x_1) \\ 3a_nx_{n+1}^2 + 2b_nx_{n+1} + c_n &= f'(x_{n+1}). \end{aligned}$$

Di queste tre categorie di condizioni proposte, se ne sceglie solo una, ogni volta che si calcola una spline. Non è che queste condizioni *escano dal cilindro*, cioè non sono pensate *a caso*, ma sono state studiate al fine di avere garanzie sulla convergenza delle spline cubiche. In effetti, se utilizziamo una delle spline cubiche soddisfacenti a una di queste tre condizioni, abbiamo la garanzia che la funzione converga rapidamente quanto la spline lineare.

L'obiettivo dietro a tutta la teoria che abbiamo presentato, però, era cercare di superare il tasso della convergenza delle spline lineari. In effetti, se usiamo le condizioni (2) o (3), abbiamo la garanzia, ammesso di avere una funzione abbastanza regolare ($f \in \mathrm{C}^{(4)}([a,b])$), di avere una convergenza rapida in norma infinito non solo della funzione, ma anche delle derivate! Vale infatti il seguente teorema. Sia $h_i = x_{i+1} - x_i$, il passo legato all'intervallo i-esimo. Sia poi

$$h = \max_{1 \le i \le n} h_i.$$

Se vigono le condizioni aggiuntive (2) o (3), ovvero spline *not-a-knot* o *vincolata*, se $f \in \mathrm{C}^{(4)}([a,b])$, e se

$$\frac{h}{h_i} \le \gamma < \infty,$$

allora

$$\left\| f^{(p)} - S_3^{(p)} \right\|_\infty = \mathcal{O}(h^{4-p}), \quad h \to 0, p = 0, 1, 2, 3.$$

Spendiamo un po' di tempo per commentare questo teorema. Prima di tutto, commentiamo l'ipotesi sul massimo passo h:

$$\frac{h}{h_i} \le \gamma < \infty.$$

Scrivere questo significa scrivere

$$h \le \gamma h_i,$$

dove si richiede che γ sia minore di infinito e, conseguentemente, h_i diverso da zero. Questa scrittura è semplicemente un modo di dire che **la griglia deve essere più o meno uniforme**: non dobbiamo avere dei passi h_i troppo piccoli o tantomeno il massimo passo h non deve essere troppo grosso rispetto agli altri. Se vige questo, abbiamo che la p-esima derivata della spline converge uniformemente alla funzione. Questo è più di quanto avevamo visto finora: fino a questo momento ci eravamo posti domande sulla convergenza della funzione, non della derivata; quando abbiamo un fenomeno come quello appena osservato, possiamo parlare di **simultanea approssimazione**. Da un lato, questa ci permette una convergenza molto rapida della funzione ($p = 0$), che sarà $\mathcal{O}(n^4)$ in virtù di quanto abbiamo appena detto; *simultaneamente*, anche le derivate convergeranno.

Tutto è bene quel che finisce bene, ma a questo punto dobbiamo spezzare una lancia a favore della interpolazione polinomiale. Infatti, qui abbiamo questo $\mathcal{O}(n^{4-p})$, con quel 4 che in qualche modo limita il tasso di convergenza. Infatti, se avessimo per esempio una funzione con 12 derivate continue, quindi $\mathrm{C}^{(12)}([a,b])$, il tasso di convergenza sarebbe sempre lo stesso di quello di una funzione $\mathrm{C}^{(4)}([a,b])$. Questo è un limite legato alle spline cubiche, e viene detto **fenomeno della saturazione**. In questo senso, i polinomi interpolanti, se usati adeguatamente[3], come avevamo visto, permettono di ottenere una convergenza uniforme tipo

$$\|f - p_n\|_\infty = \mathcal{O}\left(\frac{\log n}{n^k}\right),$$

dove k è l'ordine della derivata massima continua su $[a,b]$: qui, non abbiamo nessun fenomeno di saturazione!

3.4 Cenni al metodo dei trapezi

Un argomento che non verrà approfondito in queste lezioni, ma che vale la pena di menzionare in qualità di *applicazione delle spline lineari* è il **metodo dei trapezi** per il calcolo dell'integrale di una funzione. L'idea di questo metodo è molto semplice: approssimare una funzione mediante una spline lineare, e quindi calcolare l'area di ciascun trapezio di base maggiore B, base minore b e altezza h mediante la solita formuletta analitica

[3] con gli opportuni nodi di interpolazione

$$A = \frac{B+b}{2}h.$$

In MATLAB®, il comando `trapz` permette di realizzare questa operazione senza doverla fare *a mano*. Al fine di mostrarlo, proponiamoci di calcolare un integrale molto semplice, del quale conosciamo la soluzione analitica[4]:

$$\int_2^3 x^2\,\mathrm{d}x = \frac{x^3}{3}\Big|_2^3 = \frac{27}{3} - \frac{8}{3} = \frac{19}{3}.$$

Si può per esempio implementare il seguente script, e vedere come va:

```
clear
close all
clc

Nnodi=10; % numero di nodi di interpolazione/integrazione
a=2; b=3; % estremi di interpolazione/integrazione
x = linspace(a,b,Nnodi); % usiamo (non e' obbligatorio) nodi equispaziati
f = x.^2;
Integrale = trapz(x,f);
Integrale_esatto = 19/3;
errrel = abs(Integrale-Integrale_esatto)/Integrale_esatto
```

L'errore relativo `errrel` dovrebbe essere dell'ordine di 10^{-4} usando 10 nodi; questa non è una tecnica molto raffinata, ma di sicuro può essere un punto di partenza nel caso capitassero problemi di integrazione semplici da risolvere.

A Esempio di uso di spline lineari

Viene ora riportato uno script MATLAB®che permette di generare, una volta fissato il numero di intervalli, la spline lineare interpolante una funzione nota. Lanciando questo codice per diversi intervalli, è possibile apprezzare la convergenza uniforme di queste funzioni.

```
clear
close all
clc

nvet = [1 2 3 4 5 6 7 8 9 10 11 12]; % specificare i numeri di nodi (-1) della partizione
a = -5; % estremo inferiore intervallo
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare la spline

figure(1)
set(gcf,'Position',[381 175 1208 753])

for indn = 1:length(nvet)
    n = nvet(indn);
    x = linspace(a,b,n+1); % in questo esercizio definisco i nodi di interpolazione tra loro ↩
        equispaziati
    S = interp1(x,f(x),z); % valuto la spline lineare interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,S,'r') % blu, curva originale; cerchietti neri, nodi di ↩
        interpolazione; rosso, spline
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-S)/norm(f(z));
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

[4] per quanto la tentazione di provare da subito la Gaussiana sia forte, eh?

B Esempio di uso di spline cubiche

Viene ora riportato uno script MATLAB® che permette di generare, una volta fissato il numero di intervalli, la spline cubica interpolante una funzione nota. Lanciando questo codice per diversi intervalli, è possibile apprezzare la convergenza uniforme di queste funzioni.

```
clear
close all
clc

nvet = [1 2 3 4 5 6 7 8 9 10 11 12]; % specificare i numeri di nodi (-1) della partizione
a = -5; % estremo inferiore intervallo
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare la spline

figure(1)
set(gcf,'Position',[381 175 1208 753])

for indn = 1:length(nvet)
    n = nvet(indn);
    x = linspace(a,b,n+1); % in questo esercizio definisco i nodi di interpolazione tra loro ↩
        equispaziati
    S = spline(x,f(x),z); % valuto la spline cubica interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,S,'r') % blu, curva originale; cerchietti neri, nodi di ↩
        interpolazione; rosso, spline
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-S)/norm(f(z));
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

C Approfondimenti sulle spline

Questa appendice si pone l'obiettivo di presentare alcuni approfondimenti e riflessioni riguardanti le spline, con particolare attenzione verso le spline cubiche. Per questo motivo, è consigliata la lettura esclusivamente in caso di interesse verso l'argomento, in quanto si prenderanno in considerazione dettagli che possono essere complicati e la cui comprensione non è fondamentale.

C.1 Quando abbiamo pochi nodi: spline, polinomi interpolanti, o *entrambi*?

Al fine di comprendere queste tecniche numeriche, è opportuno applicarle a casi *semplici*, e quindi, per poter capire bene cosa sta succedendo, a griglie di interpolazione con pochi nodi. Il minimo sindacale ovviamente sarebbe usare 2 nodi: non posso ottenere alcuna informazione sull'andamento di una funzione interpolando solo in un punto!

Parlando di spline cubiche interpolanti *not-a-knot*, voglio proporre una frase un po' forte.

> «Quando abbiamo un numero di nodi minore o uguale a 4, non ha senso distinguere polinomi interpolanti e spline.»

Perché mai? Beh, qualche lezione fa, si era detto che

> «Assegnati $n+1$ dati (x_i, y_i), $i = 1, \ldots, n+1$, esiste uno e un solo polinomio $p_n(x)$ di grado minore o uguale a n interpolante i dati assegnati, ovvero soddisfacente le condizioni di interpolazione.»

Ovvero, il famoso teorema di esistenza e unicità. Questo ci dice che il polinomio interpolante questi punti ha grado 3. Da un altro punto di vista, se immaginiamo di usare 4 nodi di interpolazione e di cercare una spline cubica *not-a-knot*, dovremo dividere il nostro intervallo in tre sotto-intervalli, e imporre:

- quattro condizioni di continuità;
- due condizioni di continuità della derivata prima della spline cubica;
- due condizioni di continuità della derivata seconda della spline cubica;
- **due condizioni di continuità della derivata terza della spline cubica** (*not-a-knot*).

Come avevamo scritto nel testo, le condizioni *not-a-knot* fanno sì che il primo e il secondo intervallo, nonché il penultimo e l'ultimo, abbiano polinomi uguali tra loro, quindi con coefficienti assolutamente identici. Dire che i coefficienti sono identici, è come dire che abbiamo un unico polinomio definito su tutti e tre i sotto-intervalli.

Questa dimostrazione *astratta* per quanto corretta può essere rafforzata da un esperimento numerico. A tal fine, si propone un codice che interpola una certa funzione su quattro nodi disposti in modo casuale mediante polinomio interpolante e mediante spline cubica; studiando l'errore relativo, si può vedere che è comparabile alla precisione di macchina e che, quindi, i due procedimenti, con 4 nodi, **sono esattamente uguali**. Il seguente codice MATLAB®permette di dimostrare questo fatto mediante un esperimento numerico.

```
clear
close all
clc

x=[0.1 0.5 1 3]; % definizione di alcuni nodi disposti a caso
y=cos(x); % usiamo come funzione da interpolare una funzione a caso

z = linspace(min(x),max(x),1001); % definisco la griglia fine

% Interpolo mediante pline
S = spline(x,y,z);

% Interpolo mediante polinomio interpolante
c = polyfit(x,y,length(x)-1);
p = polyval(c,z);

% Calcolo l'errore in norma infinito tra le due rappresentazioni
errrel = norm(S-p,inf)/norm(p,inf)
```

Possiamo invece dire qualcosa riguardo al caso con 2 e 3 nodi? Sì: sono identici a quello a 4 nodi, e per lo stesso motivo. In effetti, quando abbiamo 2 nodi, dire che usiamo un polinomio interpolante di grado 1 o dire che usiamo una spline lineare è esattamente la stessa cosa: le due curve sono coincidenti. Allo stesso modo, il caso con 3 nodi, produrrà, con i comandi `spline` e `polyfit/polyval`, esattamente gli stessi risultati, e per la stessa spiegazione che abbiamo fornito. Quindi, perché le spline siano un qualcosa di effettivamente diverso dal polinomio interpolante, è necessario aver a che fare almeno con 5 nodi di interpolazione.

C.2 Ma come fa MATLAB a calcolare le spline?!

Volendo essere **davvero** convinti riguardo a ciò che è stato scritto nel testo, in questa appendice studiamo ora uno schema numerico che permette di calcolare i coefficienti delle spline per un caso molto semplice: una funzione interpolata in 5 nodi (che, come abbiamo imparato nella precedente sottosezione, è il primo caso che abbia davvero senso studiare mediante le spline). In questa sezione, quindi, prima proporremo la formulazione teorica, e poi un'implementazione dell'algoritmo atto a ricavare i coefficienti delle spline su 5 nodi. Da un punto di vista concettuale, questo è esattamente ciò che fa MATLAB®(che però sarà ottimizzato in vari modi dal punto di vista della programmazione).

Il primo passo è ricordare l'espressione delle condizioni di interpolazione (3); nell'esercizio che stiamo svolgendo, abbiamo 5 nodi, e quindi le condizioni si possono scrivere esplicitamente come

$$
\begin{aligned}
a_1x_1^3 + b_1x_1^2 + c_1x_1 + d_1 &= y_1\\
a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 &= y_2\\
a_3x_3^3 + b_3x_3^2 + c_3x_3 + d_3 &= y_3\\
a_4x_4^3 + b_4x_4^2 + c_4x_4 + d_4 &= y_4\\
a_4x_5^3 + b_4x_5^2 + c_4x_5 + d_4 &= y_5.
\end{aligned}
$$

Come spiegheremo in dettaglio più avanti[5], queste espressioni si possono scrivere in termini di un prodotto matriciale riga per colonna come

$$
\begin{bmatrix}
x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_5^3 & x_5^2 & x_5 & 1
\end{bmatrix}
\begin{bmatrix}
a_1\\ b_1\\ c_1\\ d_1\\ a_2\\ b_2\\ c_2\\ d_2\\ a_3\\ b_3\\ c_3\\ d_3\\ a_4\\ b_4\\ c_4\\ d_4
\end{bmatrix}
=
\begin{bmatrix}
y_1\\ y_2\\ y_3\\ y_4\\ y_5
\end{bmatrix}.
$$

Infatti, si può vedere *facilmente* che, calcolando i prodotti riga per colonna, si ottengono le cinque equazioni scritte prima.

Proseguiamo a questo punto con le condizioni di continuità delle spline; partendo da (4) possiamo scrivere esplicitamente, portando tutte le espressioni al membro sinistro,

$$
\begin{aligned}
a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 - (a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2) &= 0\\
a_2x_3^3 + b_2x_3^2 + c_2x_3 + d_2 - (a_3x_3^3 + b_3x_3^2 + c_3x_3 + d_3) &= 0\\
a_3x_4^3 + b_3x_4^2 + c_3x_4 + d_3 - (a_4x_4^3 + b_4x_4^2 + c_4x_4 + d_4) &= 0,
\end{aligned}
$$

che si scrivono, in forma matriciale,

$$
\begin{bmatrix}
x_2^3 & x_2^2 & x_2 & 1 & -x_2^3 & -x_2^2 & -x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & -x_3^3 & -x_3^2 & -x_3 & -1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 & -x_4^3 & -x_4^2 & -x_4 & -1
\end{bmatrix}
\begin{bmatrix}
a_1\\ b_1\\ c_1\\ d_1\\ a_2\\ b_2\\ c_2\\ d_2\\ a_3\\ b_3\\ c_3\\ d_3\\ a_4\\ b_4\\ c_4\\ d_4
\end{bmatrix}
=
\begin{bmatrix}
0\\ 0\\ 0
\end{bmatrix}.
$$

[5] nell'ambito dello studio dei sistemi lineari

Seguono quindi le condizioni di continuità della derivata prima, che si scrivono per esteso, a partire da (5),

$$\begin{aligned}
3a_1x_2^2 + 2b_1x_2 + c_1 - (3a_2x_2^2 + 2b_2x_2 + c_2) &= 0\\
3a_2x_3^2 + 2b_2x_3 + c_2 - (3a_3x_3^2 + 2b_3x_3 + c_3) &= 0\\
3a_3x_4^2 + 2b_3x_4 + c_3 - (3a_4x_4^2 + 2b_4x_4 + c_4) &= 0
\end{aligned}$$

e quindi, in forma matriciale, come

$$\begin{bmatrix}
3x_2^2 & 2x_2 & 1 & 0 & -3x_2^2 & -2x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 3x_3^2 & 2x_3 & 1 & 0 & -3x_3^2 & -2x_3 & -1 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3x_4^2 & 2x_4 & 1 & 0 & -3x_4^2 & -2x_4 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
a_1\\ b_1\\ c_1\\ d_1\\ a_2\\ b_2\\ c_2\\ d_2\\ a_3\\ b_3\\ c_3\\ d_3\\ a_4\\ b_4\\ c_4\\ d_4
\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}.$$

Seguendo un procedimento analogo, è possibile scrivere anche le condizioni sulla derivata seconda (6):

$$\begin{aligned}
6a_1x_2 + 2b_1 - (6a_2x_2 + 2b_2) &= 0\\
6a_2x_3 + 2b_2 - (6a_3x_3 + 2b_3) &= 0\\
6a_3x_4 + 2b_3 - (6a_4x_4 + 2b_4) &= 0,
\end{aligned}$$

che si possono scrivere in forma matriciale come segue:

$$\begin{bmatrix}
6x_2 & 2 & 0 & 0 & -6x_2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 6x_3 & 2 & 0 & 0 & -6x_3 & -2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6x_4 & 2 & 0 & 0 & -6x_4 & -2 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
a_1\\ b_1\\ c_1\\ d_1\\ a_2\\ b_2\\ c_2\\ d_2\\ a_3\\ b_3\\ c_3\\ d_3\\ a_4\\ b_4\\ c_4\\ d_4
\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}.$$

Infine, per quanto riguarda le condizioni *not-a-knot* si ha, a partire da

$$\begin{aligned}
a_1 - a_2 &= 0\\
a_3 - a_4 &= 0,
\end{aligned}$$

la scrittura matriciale

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Impilando tutte queste matrici, è possibile trovare un'equazione matriciale nella forma

$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_5^3 & x_5^2 & x_5 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 & -x_2^3 & -x_2^2 & -x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & -x_3^3 & -x_3^2 & -x_3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 & -x_4^3 & -x_4^2 & -x_4 & -1 \\ 3x_2^2 & 2x_2 & 1 & 0 & -3x_2^2 & -2x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3x_3^2 & 2x_3 & 1 & 0 & -3x_3^2 & -2x_3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3x_4^2 & 2x_4 & 1 & 0 & -3x_4^2 & -2x_4 & -1 & 0 \\ 6x_2 & 2 & 0 & 0 & -6x_2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6x_3 & 2 & 0 & 0 & -6x_3 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6x_4 & 2 & 0 & 0 & -6x_4 & -2 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Segue quindi uno script che implementa tutte le formule appena ricavate al fine di ricavare i coefficienti e valutare le spline.

```
clear
close all
clc

% definiamo alcuni nodi in cui andare a interpolare
x1=0.1;
x2=0.5;
x3=1;
x4=3;
x5=4;

x = [x1 x2 x3 x4 x5];
y=cos(x); % usiamo una funzione a caso

n=length(x)-1;
b = zeros(4*n,1);

% Matrice delle condizioni di interpolazione
AI = [x1^3  x1^2    x1     1     0     0     0     0     0     0     0     0     0     0     0  0;
         0     0     0     0  x2^3  x2^2    x2     1     0     0     0     0     0     0     0  0;
         0     0     0     0     0     0     0     0  x3^3  x3^2    x3     1     0     0     0  0;
         0     0     0     0     0     0     0     0     0     0     0     0  x4^3  x4^2    x4  1;
         0     0     0     0     0     0     0     0     0     0     0     0  x5^3  x5^2    x5  1];
b(1:n+1)=y;
```

```
% Matrice delle condizioni di continuita`
AC = [x2^3  x2^2    x2     1 -x2^3 -x2^2   -x2    -1     0     0     0     0     0     0     0  0;
         0     0     0     0  x3^3  x3^2    x3     1 -x3^3 -x3^2   -x3    -1     0     0     0  0;
         0     0     0     0     0     0     0     0  x4^3  x4^2    x4     1 -x4^3 -x4^2   -x4 -1];

% Matrice delle condizioni di continuita` derivata prima
AD1 = [3*x2^2 2*x2^1  1   0 -3*x2^2 -2*x2^1  -1 0       0       0  0 0       0       0   0   0;
           0     0  0   0  3*x3^2  2*x3^1   1 0 -3*x3^2 -2*x3^1 -1 0       0       0   0   0;
           0     0  0   0       0       0   0 0  3*x4^2  2*x4^1  1 0 -3*x4^2 -2*x4^1  -1   0];

% Matrice delle condizioni di continuita` derivata seconda
AD2 = [6*x2     2  0     0    -6*x2   -2     0  0     0     0     0  0     0     0  0  0;
          0     0  0     0     6*x3    2     0  0 -6*x3    -2     0  0     0     0  0  0;
          0     0  0     0     0       0     0  0  6*x4     2     0  0 -6*x4    -2  0  0];

% Matrice delle condizioni not-a-knot
ANAK = [1     0     0     0    -1     0     0     0  0     0     0     0     0     0     0;
        0     0     0     0     0     0     0     0  0     0     0     1     0     0     0    -1];

% Assemblo la matrice finale impilando le varie matrici
A = [AI;
     AC;
     AD1;
     AD2;
     ANAK
    ];

% Calcolo i coefficienti risolvendo il sistema lineare
Coeff = A\b;

% I coefficienti sono ordinati come segue:
% [a1,b1,c1,d1,a2,b2,c2,d2,a3,b3,c3,d3,a4,b4,c4,d4]
Coeff1=Coeff(1:4);
Coeff2=Coeff(4+[1:4]);
Coeff3=Coeff(8+[1:4]);
Coeff4=Coeff(12+[1:4]);

% Per semplificare, definisco 4 vettori di z, uno per ciascun intervallo.
z1 = 0.1:0.01:0.5;
z2 = 0.5:0.01:1;
z3 = 1:0.01:3;
z4 = 3:0.01:4;

% Calcolo le spline usando la formula e i relativi coefficienti
S3_1 = Coeff1(1)*z1.^3 + Coeff1(2)*z1.^2 + Coeff1(3)*z1 + Coeff1(4);
S3_2 = Coeff2(1)*z2.^3 + Coeff2(2)*z2.^2 + Coeff2(3)*z2 + Coeff2(4);
S3_3 = Coeff3(1)*z3.^3 + Coeff3(2)*z3.^2 + Coeff3(3)*z3 + Coeff3(4);
S3_4 = Coeff4(1)*z4.^3 + Coeff4(2)*z4.^2 + Coeff4(3)*z4 + Coeff4(4);

z = [z1 z2 z3 z4];
S3 = [S3_1 S3_2 S3_3 S3_4];

% Calcolo le spline con le funzioni MATLAB, per verificare la correttezza
S=spline(x,y,z);

figure(3289)
grid on
hold on
box on
plot(z,S3,z,S,'--')

errrel=norm(S-S3)/norm(S) % l'errore relativo e` nullo: i procedimenti sono uguali!
```

Questo esempio permette di capire davvero quanto sia complicato calcolare una spline anche per il caso più semplice possibile: per trovare i coefficienti relativi all'interpolazione di una funzione su 5 nodi è necessario risolvere un sistema 16×16! Inoltre, questo script di esempio permette di dimostrare e chiarire ulteriormente alcuni concetti. Se consideriamo i vettori dei vari coefficienti `Coeff1`, `Coeff2`, `Coeff3`, `Coeff4`, ciascuno relativo a uno dei quattro sottointervalli, possiamo farli stampare sulla command window (ciascuno sarà un vettore colonna contenente, su ciascuna riga, a_j, b_j, c_j, d_j, dove j è l'indice dell'intervallo), ottenendo

```
>> [Coeff1 Coeff2 Coeff3 Coeff4]
ans =

    0.1574    0.1574    0.1140    0.1140
   -0.6753   -0.6753   -0.5450   -0.5450
    0.0628    0.0628   -0.0675   -0.0675
    0.9953    0.9953    1.0387    1.0387
```

>>

Questo è molto interessante: i coefficienti della prima e della seconda spline sono uguali tra loro, così come quelli della terza e della quarta. Questa è la conseguenza della condizione *not-a-knot*, come già dimostrato formalmente nel testo!