

Può succedere che non è prevista una funzione per scrivere il polinomio d'interpolazione in forma di Newton

Osservazione 1.1. Supponiamo che siano dati i punti $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^2$ con x_0, x_1, \dots, x_n distinti. I numeri y_0, y_1, \dots, y_n possono sempre essere interpretati come i valori in x_0, x_1, \dots, x_n di una qualche funzione $f : [a, b] \rightarrow \mathbb{R}$ definita su un qualche intervallo $[a, b]$ che contiene i nodi x_0, \dots, x_n . Dunque, ha perfettamente senso parlare di forma di Newton del polinomio d'interpolazione dei dati $(x_0, y_0), \dots, (x_n, y_n)$:

Basta infatti immaginarsi una qualche $f(x)$ tale che $f(x_i) = y_i \quad \forall i = 0, \dots, n$
e il gioco è fatto

> ALGORITMO DI VALUTAZIONE DEL POLINOMIO D'INTERPOLAZIONE IN UN PUNTO

- Algoritmo basato sulla forma di Newton
- Algoritmo efficiente dal punto di vista computazionale (ne calcoleremo il costo)

Sia $f : [a, b] \rightarrow \mathbb{R}$, siano $x_0, x_1, \dots, x_n \in [a, b]$ punti distinti e sia $t \in \mathbb{R}$. Si vuole costruire un algoritmo per calcolare $p(t)$, dove $p(x)$ è il polinomio d'interpolazione di $f(x)$ sui nodi x_0, x_1, \dots, x_n .

Per semplicità, descriviamo l'algoritmo nel caso $n=3$, cosicché per la forma di Newton (N) è

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2). \quad (1.12)$$

L'algoritmo è composto da 2 parti

- PARTE 1: Indipendente dal punto t in cui devo valutare $p(x)$.

Consiste nel calcolo delle differenze divise rosse, calcolo che viene effettuato con la seguente tabella

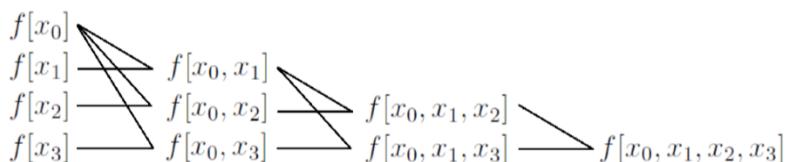


Tabella 1.1: Tabella delle differenze divise.

- PARTE 2: Una volta calcolate le differenze divise rosse, per calcolare $p(t)$ si usa un metodo noto come algoritmo di Ruffini-Horner:
si scrive $p(t)$ nella forma



$$p(t) = f[x_0] + (t - x_0) \left(f[x_0, x_1] + (t - x_1) \left(f[x_0, x_1, x_2] + (t - x_2) f[x_0, x_1, x_2, x_3] \right) \right)$$

e si pone

$$\begin{aligned} h_3 &= f[x_0, x_1, x_2, x_3], \\ h_2 &= f[x_0, x_1, x_2] + (t - x_2)h_3, \\ h_1 &= f[x_0, x_1] + (t - x_1)h_2, \\ h_0 &= f[x_0] + (t - x_0)h_1. \end{aligned}$$

h_0 è proprio $p(t)$.

Valutiamo il costo computazionale dell'algoritmo.

■ PARTE 1:

Si devono calcolare $6 = \frac{n(n+1)}{2}$ elementi della tabella (tutti meno quelli della prima colonna che sono noti).

Il numero $\frac{m(m+1)}{2}$ di elementi da calcolare coincide con il numero di elementi della parte triangolare inferiore (inclusa la diagonale) di una matrice triangolare ~~inferiore~~ $n \times n$, ossia $\frac{n^2-n}{2} + n = \frac{n(n+1)}{2}$ ovvero $1+2+\dots+n = \frac{n(n+1)}{2}$.

Per il calcolo di ciascun elemento occorrono 2 sottrazioni e 1 divisione, per cui in totale

- $n(n+1)$ sottrazioni,
- $\frac{n(n+1)}{2}$ divisioni.

$$\rightarrow m(m+1)A + \frac{m(m+1)}{2}D$$

■ PARTE 2:

Si devono calcolare h_2, h_1, h_0 ($h_3 = h_n$ non va calcolato perché è una differenza divisa che è già stata calcolata nella parte 1). Per il calcolo di ciascun h_i , con $i = m-1, \dots, 0$ sono richieste 1 sottrazione, 1 moltiplicazione e 1 addizione, in totale:

- n sottrazioni,
- n moltiplicazioni,
- n addizioni.

$$\rightarrow 2mA + mM$$

Notiamo che una sottrazione costa come un'addizione per il computer: la sottrazione $\xi - \eta$ coincide con l'addizione $\xi + (-\eta)$ e per il computer mettere un segno meno davanti a η non costa niente (non deve fare nessuna operazione). Considereremo pertanto le sottrazioni come se fossero delle addizioni. Dunque, indicando con A, M e D rispettivamente le addizioni, le moltiplicazioni e le divisioni,

- il costo del calcolo delle differenze divise è $n(n+1)A + \frac{n(n+1)}{2}D$,
- il costo del calcolo di h_{n-1}, \dots, h_0 è $2nA + nM$,
- il costo complessivo dell'algoritmo è

$$c(n) = (n^2 + 3n)A + nM + \left(\frac{n^2}{2} + \frac{n}{2}\right)D \approx n^2 A + \frac{n^2}{2} D,$$

per un totale di $\frac{3}{2}n^2 + \frac{9}{2}n \approx \frac{3}{2}n^2$ operazioni.