Calcolo Scientifico - Laurea triennale Informatica
APPROSSIMAZIONE di DATI e FUNZIONI

prof.ssa Rossana Vermiglio, dott. Dimitri Breda
Dipartimento di Matematica e Informatica
Università degli Studi di Udine
vermiglio,dbreda@dimi.uniud.it
http://www.dimi.uniud.it/∼rossana,dbreda

Indice

1 Introduzione all'interpolazione **2**

2 Polinomi e interpolazione **3**
2.1 Polinomio di Lagrange 3
2.2 Polinomio di Newton 4
2.3 Differenze divise e polinomio di Newton 5
2.4 Errore interpolazione e convergenza 5

3 Polinomio di Hermite **6**

4 Polinomi a tratti e interpolazione **7**

5 Spline e interpolazione **7**
5.1 Convergenza delle splines interpolanti 11
5.2 Splines parametriche 11
5.3 B-splines 12

6 Curve di Bézier e B-splines parametriche. **13**

7 Polinomi trigonometrici e interpolazione. FFT **14**

8 MATLAB **16**
8.1 Interpolazione 16

9 Introduzione alla miglior approssimazione: minimi quadrati **18**

10 Esercizi svolti **20**
10.1 Interpolazione 20
10.2 Minimi quadrati 35

1 Introduzione all'interpolazione

Il problema dell'**interpolazione** può essere così descritto:

Assegnati i dati $(t_i, y_i), i = 0, \ldots, m$, con $t_i \neq t_j, i \neq j$, determinare una funzione φ tale che

$$\varphi(t_i) = y_i, i = 0, \ldots, m \tag{1}$$

La funzione φ si chiama **funzione interpolante** o semplicemente **interpolante**, i punti distinti $t_i, i = 0, \ldots, m$ sono i **nodi** dell'interpolazione e le relazioni (1) definiscono le **condizioni di interpolazione**.

In alcuni contesti applicativi si possono imporre ulteriori condizioni quali, per esempio, avere una certa pendenza in alcuni punti assegnati, proprietà di liscezza, monotonia e convessità.

Se i dati risultano da un campionamento di una funzione f, i.e. $y_i = f(t_i), i = 0, \ldots, m$, l'interpolazione permette di approssimare la funzione f "difficile" con una funzione φ "semplice", i.e. meno costosa computazionalmente.

L'interpolazione viene proposta per ricostruire curve sufficientemente liscie che passano attraverso dei dati discreti, per ottenere dei valori in punti t diversi dai dati osservati. Infine la tecnica dell'interpolazione trova applicazione nell' approssimazione di derivate ed integrali.

L'interpolazione non è una tecnica appropriata quando i dati del problema sono affetti da errori significativi: in questo caso la tecnica della miglior approssimazione è da preferire.

Come scegliere la funzione φ?

La scelta più opportuna è suggerita dal contesto applicativo e dalle proprietà che che si vogliono ottenere, come mantenere la periodicità o la monotonia dei adati, proprietà di lisciezza, accuratezza etc.

Una caratteristica fondamentale della funzione interpolante consiste nell'essere rappresentabile come combinazione lineare di **funzioni base** $\phi_0(t), \ldots, \phi_m(t)$ "facilmente calcolabili"i.e.

$$\varphi(t) = \sum_{j=0}^{m} x_i \phi_i(t). \tag{2}$$

Usando la (2), le condizioni di interpolazione (1) portano alle seguenti equazioni linerari per la determinazione dei coefficienti x_i della combinazione lineare (2)

$$\varphi(t_i) = \sum_{j=0}^{m} x_j \phi_j(t_i) = y_i, i = 0, \ldots, m,$$

equivalenti al sistema lineare di dimensione $m + 1$

$$Ax = y \tag{3}$$

dove $a_{i,j} = \phi_j(t_i), i, j = 0, \ldots, m,$, $y = (y_0, \ldots, y_m)^T, x = (x_0, \ldots, x_m)^T$.

Le propriet'à e la struttura della matrice A dipendono sia dalla scelta delle funzioni base $\phi_j, j = 0, \ldots, m$ che dai nodi di interpolazione.

Le famiglie di funzioni φ solitamente usate sono:

- Polinomi → Interpolazione polinomiale
- Polinomi a tratti → Interpolazione polinomiale a tratti
- Polinomi trigonometrici → Interpolazione trigonometrica
- Funzioni razionali → Interpolazione razionale

2 Polinomi e interpolazione

Scegliamo le funzioni $\phi_i(t) = t^i, i = 0, \ldots, m$ (**monomi**) come base di rappresentazione del generico polinomio di grado m. I coefficienti del polinomio interpolante $p_m(t) = x_0 + x_1 t + \ldots + x_m t^m$, sono soluzione del sistema (3), dove la matrice è definita come segue

$$A = \begin{pmatrix} 1 & t_0 & \ldots & t_0^m \\ 1 & t_1 & \ldots & t_1^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_1 & \ldots & t_1^m \end{pmatrix}. \tag{4}$$

Tale matrice si chiama **di Vandermonde** e risulta non singolare se i nodi di interpolazione sono distinti.

Possiamo pertanto concludere che, se i nodi d'interpolazione sono distinti, esiste ed è unico il polinomio $p_m(t)$ di grado al più m che interpola i dati $(t_i, y_i), i = 0, \ldots, m$.

Per determinare numericamente i coefficienti del polinomio, possiamo calcolare la soluzione del sistema lineare con $O(m^3)$ operazioni. Ma la matrice di Vandermonde risulta malcondizionata specialmente per m grandi e pertanto i coefficienti x_i possono risultare poco accurati.

Il condizionamento può essere leggermente migliorato considerando le seguenti funzioni base

$$\phi_i(t) = (\frac{t-c}{d})^i, i = 0, \ldots, m \tag{5}$$

con $c = (t_0 + t_m)/2$ ('shift') e $d = (t_m - t_0)/2$ ('scaling')

Tali operazioni di shift e scaling aiutano ad evitare overflow e/o underflow.

Il problema del malcondizionamento ed anche il costo computazionale del calcolo dei coefficienti possono essere migliorati cercando altre basi per la rappresentazione di un polinomio.

Il polinomio rimane lo stesso viene solo proposta una sua diversa rappresentazione.

2.1 Polinomio di Lagrange

Le funzioni base di Lagrange $l_i(t), i = 0, \ldots, m$ sono definite da

$$l_i(t) = \prod_{j=0, j\neq i}^{m} \frac{(t - t_j)}{(t_i - t_j)}, i = 0, \ldots, m \tag{6}$$

e verificano le relazioni $l_i(t_j) = 0$ se $i \neq j$ e $l_i(t_i) = 1$.

Il polinomio di Lagrange è facile da determinare, perhchè la matrice del sistema (**??**) risulta l'indentità e pertanto il polinomio interpolante nella rappresentazione di Lagrange, i.e. **polinomio di Lagrange** risulta

$$p_m(t) = \sum_{i=0}^{m} l_i(t) y_i. \tag{7}$$

La valutazione del polinomio in un punto risulta meno efficiente rispetto alla rappresentazione con i monomi, dove si ha a disposizione l'algoritmo di Horner.

Tale rappresentazione ha notevole importanza teorica nello studio del condizionamento, nell'analisi della convergenza e nelle applicazioni per la derivazione numerica con tecniche pseudospettrali e nell'integrazione numerica.

2.2 Polinomio di Newton

Le funzioni base di Newton $\phi_i(t), i = 0, \ldots, m$ sono definite da

$$\phi_0(t) = 1 \phi_i(t) = \prod_{j=0}^{i-1} (t - t_j), i = 1, \ldots, m. \tag{8}$$

La matrice del sistema (3) risulta triangolare inferiore e pertanto i coefficienti x_i della rappresentazione del **polinomio di Newton**

$$p_m(t) = x_0 + x_1(t - t_0) + x_2(t - t_0)(t - t_1) + ... + x_m \prod_{j=0}^{m-1} (t - t_j) \tag{9}$$

possono essere calcolati con l'algoritmo di sostituzione in avanti con $O(m^2)$ operazioni.

Il polinomio non dipende dall'ordinamento dei nodi di interpolazione ma il condizionamento della matrice risente dall'ordine dei nodi. Spesso l'ordinamento naturale crescente non risulta il migliore ed il condizionamento spesso migliora ordinando i punti in funzione della distanza dal punto medio oppure dal punto in cui si vuole valutare il polinomio.

Tale rappresentazione del polinomio interpolante presenta notevoli vantaggi computazionali.

La valutazione del polinomio di Newton in uno o piú punti può essere ottenuta mediante una variante dell'algoritmo di Horner (esercizio).

L'aggiunta di un ulteriore nodo di interpolazione richiede il calcolo del del solo coefficiente x_{m+1} del nuovo polinomio $p_{m+1}(t)$. Si deve aggiungere un'altra riga della matrice in (3) e risolvere rispetto alla nuova incognita x_{m+1} l'ultima equazione utilizzando i valori giá calcolati $x_i, i = 0, \ldots, m$.

Per ottenere i coefficienti del polinomio di Newton c'è un'altra strada che utilizza le **differenze divise**.

2.3 Differenze divise e polinomio di Newton

Assegnati i dati $(t_i, y_i), i = 0, \ldots, m$, la **differenza divisa** $f[t_i, .., t_{i+k}]$ di ordine $k \geq 0$ viene definita ricorsivamente come segue

$$\begin{array}{l} f[t_i] = y_i \quad per \quad k = 0, i = 0, ..., m \\ f[t_i, .., t_{i+k}] = \frac{f[t_{i+1}, .., t_{i+k}] - f[t_i, .., t_{i+k-1}]}{(t_{i+k} - t_i)}, \quad k > 0, i = 0, ..., m - k. \end{array} \tag{10}$$

La differenza divisa di ordine k è invariante rispetto a qualsiasi permutazione degli argomenti.

Si puó verificare che per i coefficienti del polinomio interpolante di Newton vale $x_i = f[t_0, .., t_i], i = 0, \ldots, m$.

Il calcolo delle differenze divise utilizza una tabella e richiede $O(m^2)$ operazioni, ma è meno sensibile a underflow o overflow rispetto al calcolo della matrice triangolare.

2.4 Errore interpolazione e convergenza

Se i dati che stiamo considerndo risultano dal campionamento di una funzione, si vuole studiare quanto bene l'interpolante approssima la funzione stessa in punti diversi dai nodi. Supponiamo che la funzione $f \in C^{m+1}[a, b]$. Allora si ottiene la seguente stima puntuale

$$e_m(t) = f(t) - p_m(t) = \frac{f^{m+1}(\xi)}{(m+1)!}(t - t_0) \cdots (t - t_m). \tag{11}$$

Supponendo di conoscere una limitazione D_{m+1} della derivata f^{m+1}, si ottiene la seguente maggiorazione uniforme

$$\begin{array}{ll} max_{a \leq t_0 \leq t \leq t_m \leq b} |e_m(t)| & \leq \frac{D_{m+1}}{(m+1)!} max_{a \leq t \leq b} |(t - t_0) \cdots (t - t_m)| \\ & \leq \frac{D_{m+1}}{(m+1)!} |b - a|^{m+1}. \end{array} \tag{12}$$

Se $a = t_0$ e $b = t_m$ si ha

$$max_{a=t_0 \leq t \leq t_m=b} |e_m(t)| \leq \frac{D_{m+1}}{4(m+1)} h^{m+1}, \tag{13}$$

dove $h = max_{i=0,\ldots,m-1} |t_{i+1} - t_i|$.

Se la funzione $f \in C^{\infty}[a, b]$ possiamo cercare di analizzare la convergenza al crescere di m con tale formula. Se ne deduce che l'errore tende a zero se D_{m+1} non cresce troppo in fretta rispetto ad m. Considera come esempio positivo la funzione esponenziale.

Il polinomio interpolante oscilla tra i nodi di interpolazione e tali oscillazioni possono diventare sempre più ampie al crescere di m, come si vede analizzando la funzione di Runge interpolata su nodi equidistanti.

Esiste una scelta migliore dei nodi di interpolazione: i nodi di Chebyshev $t_i = -cos(\frac{(2i+1)\pi}{2m+2}, \ i = 0, ..., m$ relativi all'intervallo $[-1, 1]$ o l'opportuna trasformazione di questi punti nel generico intervallo $[a, b]$.

Non sono equispaziati e si addensano agli estremi. Con tali nodi ho la convergenza del polinomio interpolante alla funzione di Runge.

Se $f \in C[a,b]$ non si puó comunque garantire la convergenza del polinomio interpolante sui nodi di Chebyshev, mentre si ottiene la convergenza uniforme del polinomio interpolante sui nodi di Chebyshev per ogni $f \in C^1[a,b]$.

Utilizzando le differenze divise si ottiene la seguente espressione dell'errore

$$e_m(t) = f[t_0, \ldots, t_m, t](t-t_0)\cdots(t-t_m). \tag{14}$$

e se $f \in C^{m+1}[a,b]$ vale

$$f[t_0, \ldots, t_m, t] = \frac{f^{m+1}(\xi)}{(m+1)!}. \tag{15}$$

Ciò permette di estendere la definizione delle differenze divise di ordine k anche con nodi coincidenti come segue

$$\begin{array}{l} f[t,\ldots,t] = \frac{f^{k+1}(t)}{(k+1)!}, k \geq 1 \\ k+1 \text{v}olte \end{array} \tag{16}$$

Utilizzando la tabella delle differenza divise si possono facilmente costruire polinomi intrepolanti con condizioni anche sulle derivate.

3 Polinomio di Hermite

Nell'interpolazione di Hermite vengono assegnati nei nodi $t_i, i = 0, ..., m$ non solo i valori y_i ma anche dei valori per le derivate dy_i. Il polinomio interpolante di Hermite,H_{2m+1} verifica

$$\begin{array}{l} H_{2m+1}(t_i) = y_i, i = 0, ..., m \\ H'_{2m+1}(t_i) = dy_i, i = 0, ..., m \end{array} \tag{17}$$

Di tale polinomio si può fornire una rappresentazione tipo lagrange

$$H_{2m+1}(t) = \textstyle\sum_{i=0}^{m}(h_i(t)y_i + \bar{h}_i(t)dy_i) \tag{18}$$

dove

$$\begin{array}{l} h_i(t) = l_i^2(t)(1-2l_i'(t_i)(t-t_i)),\ i = 0, ..., m \\ \bar{h}_i(t) = l_i^2(t)(t-t_i),\ i = 0, ..., m \end{array} \tag{19}$$

Tale polinomio é unico. Se $f \in C^{2m+2}[a,b]$ si puó ricavare la seguente stima dell'errore puntuale

$$E_{2m+1}(t) = f(t) - H_{2m+1}(t) = \frac{f^{2m+2}(\xi)}{(2m+2)!}((t-t_0)\cdots(t-t_m))^2. \tag{20}$$

Dalla stima puntuale si ricava la stima uniforme dell'errore

$$max_{a\leq t_0\leq t\leq t_m\leq b}|E_{2m+1}(t)| \leq \frac{D_{2m+2}}{(2m+2)!} max_{a\leq t\leq b}((t-t_0)\cdots(t-t_m))^2 \tag{21}$$

Se $a = t_0$ e $b = t_m$ si ha

$$max_{a=t_0 \leq t \leq t_m = b} |e_m(t)| \leq \frac{D_{2m+2}(m!)^2}{16(2m+2)!} h^{2m+2} \tag{22}$$

dove $h = max_{i=0,\ldots,m-1} |t_{i+1} - t_i|$.

Utilizzando le differenze divise

$$f[t_i] = y_i, f[t_i, t_i] = dy_i, i = 0, ..., m$$

si può facilmente ottenere una rappresentazione del polinomio di Hermite tipo Newton.

Consideriamo, per esempio, il polinomio **cubico di Hermite**, H_3, che soddisfa alle seguenti condizioni di interpolazione:

$$H_3(t_0) = y_0,\ H_3(t_1) = y_1,\ H_3'(t_0) = dy_0,\ H_3'(t_1) = dy_1.$$

Si rappresenta nella forma di Newton come segue:

$$H_3(t) = f[t_0] + f[t_0, t_0](t - t_0) + f[t_0, t_0, t_1](t - t_0)^2 + f[t_0, t_0, t_1, t_1](t - t_0)^2(t - t_1)$$

4 Polinomi a tratti e interpolazione

5 Spline e interpolazione

Si consideri una suddivisione $\Delta_m = \{a = t_0 < t_2 < \cdots < t_m = b\}$ dell'intervallo $[a, b] \subset \mathbb{R}$.

Si definisce *spline di grado n* relativa alla suddivisione Δ_m la funzione $s_{n,m} : [a, b] \to \mathbb{R}$ tale che

i) $s_{n,m} \in C^{n-1}([a, b])$

ii) $s_{n,m_{|I_i}} \in \prod_n$, dove $I_i = [t_i, t_{i+1}]$, $i = 0, \ldots, m - 1$.

Una spline $s_{n,m}$ è quindi una funzione polinomiale a tratti di grado n continua su $[a, b]$ con le sue derivate fino all'ordine $n - 1$. Lo spazio delle funzioni splines $s_{n,m}$ ha dimensione $m + n$, ciò significa che la funzione $s_{n,m}$ è descritta da $m + n$ parametri. Ci restringiamo ora alle splines cubiche $s_{3,m}$, particolarmente interessanti per le applicazioni, occupandoci in particolare del problema dell'interpolazione.

In seguito, per brevità, indicheremo tali funzioni semplicemente con s_3.

Dalla definizione segue che le splines cubiche s_3 sono polinomi a tratti di grado $n = 3$ che verificano l'ulteriore condizione di regolarità : $s_3 \in C^2([a, b])$. Lo spazio ha dimensione $m + 3$ e pertanto sono necessarie $m + 3$ condizioni indipendenti per determinare univocamente una spline. Scegliendo come nodi di interpolazione gli $m + 1$ punti della suddivisione Δ_m e indicando con y_i,

$i = 0, \ldots, m$, i corrispondenti valori, si ottengono le seguenti $m + 1$ condizioni di interpolazione:

$$s_3(t_j) = y_j, \quad j = 0, \ldots, m,$$

che non sono sufficienti a determinare univocamente la spline cubica interpolante. Le due ulteriori condizioni da imporre mi permettono di definire delle particolari classi di funzioni splines:

- *naturali*: $s_3''(t_0) = s_3''(t_m) = 0$;
- *periodiche*: $s_3'(t_0) = s_3'(t_m)$ e $s_3''(t_0) = s_3''(t_m)$;
- *vincolate*: $s_3'(t_0) = y_0'$ e $s_3'(t_m) = y_m'$;
- *not-a-knot*: $s_3'''(t)$ continua in t_1 e t_{m-1}.

La scelta della particolare classe dipende dal problema e dalle proprietà che si vogliono garantire all'interpolante. Per costruire la spline cubica interpolante si può procedere come segue.

Sia $\sigma_i = s_{3|I_i} \in \prod_3$, $i = 0, \ldots, m-1$, vale pertanto $\sigma_i' \in \prod_2$ e $\sigma_i'' \in \prod_1$, ovvero la derivata seconda risulta essere una retta su ciascun intervallo I_i, $i = 0, \ldots, m-1$. Introducendo i *momenti*

$$\begin{cases} \sigma_i''(t_i) = M_i, \quad i = 0, \ldots, m-1 \\ \sigma_{m-1}''(t_m) = M_m \end{cases},$$

che sono quantità incognite, otteniamo la seguente rappresentazione

$$\sigma_i''(t) = M_{i+1}\frac{t - t_i}{h_i} - M_i\frac{t - t_{i+1}}{h_i}, \quad i = 0, \ldots, m-1$$

dove $h_i = t_{i+1} - t_i$, $i = 0, \ldots, m-1$, sono le ampiezze degli intervalli I_i. Integrando si ottiene

$$\sigma_i'(t) = M_{i+1}\frac{(t - t_i)^2}{2h_i} - M_i\frac{(t - t_{i+1})^2}{2h_i} + \alpha_i, \quad i = 0, \ldots, m-1 \tag{23}$$

ed integrando nuovamente si trovano le seguenti rappresentazioni

$$\sigma_i(t) = M_{i+1}\frac{(t - t_i)^3}{6h_i} - M_i\frac{(t - t_{i+1})^3}{6h_i} + \alpha_i(t - t_i) + \beta_i, \quad i = 0, \ldots, m-1, \tag{24}$$

dove α_i e β_i, $i = 0, \ldots, m-1$, sono costanti incognite. Imponendo le $m + 1$ condizioni di interpolazione, si trova per $i = 0, \ldots, m-1$

$$\begin{cases} \sigma_i(t_i) = y_i \\ \sigma_i(t_{i+1}) = y_{i+1} \end{cases} \Rightarrow \begin{cases} M_i\frac{h_i^2}{6} + \beta_i = y_i \\ M_{i+1}\frac{h_i^2}{6} + \alpha_i h_i + \beta_i = y_{i+1} \end{cases}.$$

Tali relazioni ci permettono di esprimere α_i e β_i, $i = 0, \ldots, m-1$, in funzione dei momenti M_i, $i = 0, \ldots, m-1$, come segue

$$\begin{cases} \alpha_i = \frac{y_{i+1} - y_i}{h_i} - (M_{i+1} - M_i)\frac{h_i}{6} \\ \beta_i = y_i - M_i\frac{h_i^2}{6} \end{cases}. \tag{25}$$

Sostituendo in (24) e (23) si ottengono rispettivamente per $i = 0, \ldots, m-1$

$$\begin{aligned} \sigma_i(t) &= M_{i+1}\frac{(t-t_i)^3}{6h_i} - M_i\frac{(t-t_{i+1})^3}{6h_i} + \\ &+ \left(\frac{y_{i+1}-y_i}{h_i} - (M_{i+1}-M_i)\frac{h_i}{6}\right)(t-t_i) + y_i - M_i\frac{h_i^2}{6} \end{aligned} \tag{26}$$

e

$$\sigma_i'(t) = M_{i+1}\frac{(t-t_i)^2}{2h_i} - M_i\frac{(t-t_{i+1})^2}{2h_i} + \frac{y_{i+1}-y_i}{h_i} - (M_{i+1}-M_i)\frac{h_i}{6}. \tag{27}$$

Imponendo infine le $m-1$ condizioni di continuità sulla derivata prima, i.e. $\sigma_i(t_{i+1}) = \sigma_{i+1}(t_{i+1})$, $i = 0, \ldots, m-2$, da (27) si ottengono le seguenti equazioni lineari per $i = 0, \ldots, m-2$

$$M_i h_i + 2M_{i+1}(h_i + h_{i+1}) + M_{i+2}h_{i+1} = 6\left(\frac{y_{i+2}-y_{i+1}}{h_{i+1}} - \frac{y_{i+1}-y_i}{h_i}\right). \tag{28}$$

Le due ulteriori equazioni si ottengono considerando la diverse classi di funzioni spline.

Per le *splines naturali* vale $M_0 = 0$ e $M_m = 0$, pertanto dalle (28) otteniamo il sistema lineare

$$HM = Y \tag{29}$$

dove $M = (M_1, \ldots, M_{m-1})^T \in \mathbb{R}^{m-1}$ è il vettore dei momenti incogniti, $Y \in \mathbb{R}^{m-1}$

$$Y = 6\begin{pmatrix} \frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0} \\ \vdots \\ \frac{y_m-y_{m-1}}{h_{m-1}} - \frac{y_{m-1}-y_{m-2}}{h_{m-2}} \end{pmatrix}.$$

è il vettore dei termini noti e la matrice $H \in \mathbb{R}^{(m-1)\times(m-1)}$ risulta

$$H = \begin{pmatrix} h_{0,1} & h_1 & & & & \\ h_1 & h_{1,2} & h_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & h_{m-3} & h_{m-3,m-2} & h_{m-2} & \\ & & & h_{m-2} & h_{m-2,m-1} & \end{pmatrix}$$

con $h_{i,j} = 2(h_i + h_j)$, $i, j = 1, \ldots, m-1$

La matrice ha struttura tridiagonale e il sistema può essere agevolmente risolto con algoritmi dedicati.

Per le *splines periodiche* si ha $\sigma_0''(t_0) = \sigma_{m-1}''(t_m) \Rightarrow M_0 = M_m$ per cui il vettore dei momenti incogniti si riduce a $M = (M_0, \ldots, M_{m-1})^T \in \mathbb{R}^m$. Inoltre $\sigma_0'(t_0) = \sigma_{m-1}'(t_m)$ implica

$$2M_0(h_0 + h_{m-1}) + M_1 h_0 + M_{m-1}h_{m_1} = 6\left(\frac{y_1-y_0}{h_0} - \frac{y_m-y_{m-1}}{h_{m-1}}\right)$$

per cui la matrice $H \in \mathbb{R}^{m\times m}$ risulta

$$H = \begin{pmatrix} h_{0,m-1} & h_0 & & & h_{m-1} \\ h_1 & h_{1,2} & h_2 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{m-3} & h_{m-3,m-2} & h_{m-2} \\ h_{m-1} & & & h_{m-2} & h_{m-2,m-1} \end{pmatrix}$$

e il vettore dei termini noti $Y \in \mathbb{R}^{m-1}$

$$Y = 6 \begin{pmatrix} \frac{y_1-y_0}{h_0} - \frac{y_m-y_{m-1}}{h_{m-1}} \\ \vdots \\ \frac{y_m-y_{m-1}}{h_{m-1}} - \frac{y_{m-1}-y_{m-2}}{h_{m-2}} \end{pmatrix}.$$

Il sistema lineare perde la struttura tridiagonale e deve essere risolto con algoritmi standard.

Per le *splines vincolate*, dalle condizioni $s_3'(t_0) = y_0'$ e $s_3'(t_m) = y_m'$ si ottengono rispettivamente

$$2M_0h_0 + M_1h_1 = 6\left(\frac{y_1 - y_0}{h_0} - y_0'\right)$$

e

$$2M_mh_{m-1} + M_{m-1}h_{m-1} = 6\left(y_m' - \frac{y_m - y_{m-1}}{h_{m-1}}\right).$$

In quest'ultimo caso il vettore dei momenti incogniti risulta $M = (M_0, \ldots, M_m)^T \in \mathbb{R}^{m+1}$, la matrice $H \in \mathbb{R}^{(m+1)\times(m+1)}$ diventa

$$H = \begin{pmatrix} 2h_0 & h_0 & & & & \\ h_0 & h_{0,1} & h_1 & & & \\ & h_1 & h_{1,2} & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{m-2} & h_{m-2,m-1} & h_{m-1} \\ & & & & h_{m-1} & 2h_{m-1} \end{pmatrix}$$

e il vettore $Y \in \mathbb{R}^{m+1}$ dei termini noti risulta

$$Y = 6 \begin{pmatrix} \frac{y_1-y_0}{h_0} - y_0' \\ \frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0} \\ \vdots \\ \frac{y_m-y_{m-1}}{h_{m-1}} - \frac{y_{m-1}-y_{m-2}}{h_{m-2}} \\ y_m' - \frac{y_m-y_{m-1}}{h_{m-1}} \end{pmatrix}.$$

Il sistema lineare conserva la struttura tridiagonale e può essere agevolmente risolto con algoritmi dedicati.

Una volta calcolati i momenti, si determinano le costanti α_i e β_i, $i = 0, \ldots, m-1$, da (25) e si ottengono finalmente le espressioni per σ_i e σ_i' rispettivamente da (26) e (27).

5.1 Convergenza delle splines interpolanti

Per l'interpolazione mediante splines cubiche vale il seguente teorema di convergenza ([Com95, Teorema 3.14]):

Teorema 1 *Sia $f \in C^4([a,b])$, $h = \max_{i=0,\ldots,m-1} |h_i|$ e si consideri una suddivisione di $[a,b]$ tale che $\frac{h}{|h_i|} \leq K$, $i = 0,\ldots,m-1$ (vale per suddivisioni equidistanti, in generale serve per evitare che un sottointervallo degeneri in un punto). Allora esiste una costante $C_k > 0$ tale che*

$$\|f - s_3\|_\infty \leq C_k K h^{4-k} \|f^{(4)}\|_\infty, \quad k = 0,1,2,3.$$

In particolare, dal precedente risultato seguono

$$\|f - s_3\|_\infty \leq \frac{7}{8} K h^4 \|f^{(4)}\|_\infty,$$

$$\|f' - s_3'\|_\infty \leq \frac{7}{4} K h^3 \|f^{(4)}\|_\infty,$$

$$\|f'' - s_3''\|_\infty \leq \frac{7}{4} K h^2 \|f^{(4)}\|_\infty$$

e

$$\|f''' - s_3'''\|_\infty \leq 2Kh \|f^{(4)}\|_\infty.$$

5.2 Splines parametriche

L'interpolazione mediante splines presenta in generale i seguenti problemi:

- l'approssimazione risultante è di buona qualità se la funzione f non presenta derivate elevate in modulo, altrimenti si potrebbe manifestare un comportamento della spline eccessivamente oscillante;
- la spline $s_{n,m}$ dipende dalla scelta del sistema di coordinate: se infatti si prova a ruotare quest'ultimo di un certo angolo, la rappresentazione della spline cambia e oscillazioni potrebbero presentarsi (vedi HW02, es.2, a.a. 2002/2003).

In particolare, nella rappresentazione di curve mediante splines, si vorrebbe che tale rappresentazione fosse indipendente dal sistema di coordinate scelto.

Una soluzione a questi problemi è rappresentata dalle *splines parametriche*: ogni componente della curva da interpolare è approssimata da una funzione spline nel seguente modo. Consideriamo una curva piana nella forma parametrica $P(t) = (x(t), y(t))$ con $t \in [0,T]$. Introduciamo una suddivisione $0 = t_0 < t_1 < \cdots < t_m = T$ di $[0,T]$ e consideriamo l'insieme dei punti punti $P_i = (x_i, y_i) = (x(t_i), y(t_i))$, $i = 0,\ldots,m$. Utilizzando i due insiemi di dati $\{t_i, x_i\}$ e $\{t_i, y_i\}$, $i = 0,\ldots,m$, possiamo costruire le due funzioni splines $s_{n,x}$ e $s_{n,y}$ rispetto alla variabile indipendente t che interpolano rispettivamente le curve $x(t)$ e $y(t)$. La curva parametrica $S_n(t) = (s_{n,x}(t), s_{n,y}(t))$ è detta *spline parametrica*.

Ovviamente sono possibili scelte differenti per la partizione di $[0, T]$. Una scelta ragionevole è la parametrizzazione basata sulla lunghezza l_i di ogni segmento $\overline{P_{i-1}P_i}$ data da

$$l_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}, \quad i = 1, \ldots, m.$$

Ponendo $t_0 = 0$ e $t_i = \sum_{k=1}^{i} l_k$ per $i = 1, \ldots, m$, ogni t_i rappresenta la lunghezza cumulativa della linea a tratti che unisce P_0 con P_i. La funzione viene chiamata *cumulative length spline* e approssima in modo soddisfacente quelle curve che presentano ampia curvatura.

Si dimostra che le splines parametriche sono geometricamente invarianti, ovvero indipendenti dal sistema di coordinate scelto per la rappresentazione.

5.3 B-splines

La funzioni B-splines rappresentano una base per le splines $s_{m,n}$.

Possono essere definite in maniera diversa attraverso la ricorsione, la convoluzione e le differenze divise.

In questa sezione presentiamo brevemente la loro definizione attraverso la ricorsione.

Come primo passo estendiamo i nodi assegnati $t_i, i = 0, \ldots, m$, scegliendo arbitrariamente degli ulteriori infiniti punti fuori dall'intervallo $[t_0, t_m]$ come segue

$$\ldots < t_{-2} < t_{-1} < t_0 < \ldots < t_m < t_{m+1} < t_{m+2} < \ldots.$$

Le B-splines di grado $n = 0$, B_i^0, sono definite da

$$B_i^0(t) = \begin{array}{ll} 1 & \text{se} t_i \leq t \leq t_{i+1} \\ 0 & \text{altrove} \end{array} \tag{30}$$

Le B-splines di grado $n > 0$, B_i^n, sono ottenute ricorsivamente attraverso le relazioni:

$$B_i^n(t) = c_i^n(t) B_i^{n-1}(t) + (1 - c_i^n(t)) B_{i+1}^{n-1}(t),$$

dove i coefficienti della combinazione sono le funzioni lineari

$$c_i^n(t) = \frac{t - t_i}{t_{i+n} - t_i}$$

.

Le funzioni $B_i^n(t)$ soddisfano le seguenti proprietà.

- Hanno supporto compatto, i.e. $B_i^n(t) = 0, t < t_i, t > t_{i+n}$. Inoltre risulta $B_i^n(t) > 0, t_i \leq t \leq t_{i+n}$.
- Per $n > 0$ le funzioni B_i^n hanno derivate continue fino all'ordine $n - 1$ e sono quindi funzioni splines.

- Le funzioni $B^n_{-n}(t), \ldots, B^n_0(t), \ldots, B^n_{m-1}$ sono linearmente indipendentiin $[t_0, t_m]$ e forniscono quindi una base per le splines $s_{n,m}$.
- Vale $\sum_{i=-\infty}^{+\infty} B^n_i(t) = 1, \forall t$ **proprietà di normalizzazione**.

Usando le B-splines come base di rappresentazione, il sistema lineare da risolvere per calcolare i coefficienti risulta non singolare e con una struttura a banda.

In questo modo sia il calcolo che la valutazione delle splines interpolanti possono essere realizzate per mezzo di metodi efficienti e stabili.

6 Curve di Bézier e B-splines parametriche.

Le B-splines parametriche e le curve di Bézier trovano applicazione nella grafica, quando i punti sono affetti da incertezza.

Siano $P_i, i = 0, ..., m,\ m+1$ punti ordinati nel piano. Il poligono che essi determinano si chiama **poligono di Bézier** o **poligono caratteristico**.

I polinomi di Bernstein $b_{k,m} \in \Pi_k$ nell'intervallo $[0, 1]$ sono definiti da

$$b_{m,k}(t) = \frac{m!}{k!(m-k)!} t^k (1-t)^{m-k}, m = 0, 1, ..,\ k = 0, ..., m.$$

e possono essere ottenuti attraverso la formula ricorsiva

$$\begin{cases} b_{m,0}(t) = (1-t)^m \\ b_{m,k}(t) = (1-t) b_{m-1,k}(t) + t b_{m-1,k-1}(t),\ k = 1, ..., m. \end{cases} \tag{31}$$

Inoltre $\{b_{m,k} | k = 0, ..., m\}$ rappresenta una base per lo spazio Π_m.

La **curva di Bézier** è definita come segue:

$$B_m(P_0, ..., P_m, t) = \sum_{k=0}^{m} b_{m,k}(t) P_k,\ t \in [0, 1]. \tag{32}$$

Tali curve possono essere ottenute anche attraverso un approccio geometrico partendo dal poligono di Bézier.

Sia $t \in [0, 1]$ un punto fissato edefiniamo i punti

$$P_{i,1}(t) = (1-t) P_i + t P_{i+1}, i = 0, ..., m-1$$

ed il nuovo poligono con m vertici che li unisce. Possiamo ripetere il precedimento generando dei nuovi vertici

$$P_{i,2}(t) = (1-t) P_{i,1} + t P_{i+1,1}, i = 0, ..., m-2$$

e il poligono che li unisce e terminando quando il poligono risultante ècostituito dai soli due vertici $P_{0,m-1}, P_{1,m-1}$.

Si può verificare che

$$P_{0,m}(t) = (1-t) P_{0,m-1} + t P_{1,m-1} = B_m(P_0, ..., P_m, t)$$

.

Ripetendo il procedimento per diversi valori di t so ricostruisce la curva di Bézier.

Osservazione: Assegnata una configurazione di nodi si possono costruire diverse curve cambiando l'ordinamento dei punti. La curva $B_m(P_0, ..., P_m, t)$ coincide con $B_m(P_m, ..., P_0, t)$ a parte l'orientamento.

Le curve di Bézier non forniscono una approssimazione sufficientemente accurata del poligono e per questa ragione sono state introdotte le **B-splines parametriche** che sono ampiamente usate nella computer graphics perchè presentano le seguenti proprietà

- la modifica di un solo vertice del poligono caratteristico induce una perturbazione locale della curva stessa attorno al vertice stesso;
- la **B-spline parametrica** approssima il poligono di controllo meglio della corrispondente curva di Bézier.

Una B-splines parametrica cubica permette di ottenere localmente una retta prendendo quattro vertici consecutivi allineati e di interpolare un punto specifico imponendo a tre vertici consecutivi del poligono di coincidere con il punto desiderato.

7 Polinomi trigonometrici e interpolazione. FFT

Consideriamo il problema di approssimare una funzione $f : [0, 2\pi] \to \mathbb{R}$ **periodica**, i.e. $f(0) = f(2\pi)$ con un polinomio trigonometrico

$$T_{2M+1}(t) = \frac{a_0}{2} + \sum_{k=1}^{M}(a_k cos(kt) + b_k sin(kt)) \tag{33}$$

Supponiamo siano noti gli $m + 1 = 2M + 1$ valori $f_k = f(t_k), k = 0, ..., m$ nei nodi $t_k = \frac{2\pi k}{m+1}, k = 0, ..., m$. Si cerca il polinomio trigonometrico interpolante

$$T_{2M+1}(t_j) = f_j, j = 0, ..., m. \tag{34}$$

Osservazione Se dati hanno m dispari, i.e. $m = 2M + 1$ si considera un polinomio trigonometrico della seguente forma

$$T_{2M+2}(t) = \frac{a_0}{2} + \sum_{k=1}^{M}(a_k cos(kt) + b_k sin(kt)) + a_{M+1} cos((M + 1)t) \tag{35}$$

Introducendo

$$e^{\imath t} = cos(t) + \imath sin(t) \tag{36}$$

con $\imath$ l'unità immaginaria, possiamo esprimere (38) come segue

$$T_{2M+1}(t) = \sum_{k=-M}^{M} c_k e^{\imath kt} \tag{37}$$

dove

$$a_k = c_k + c_{-k},\ b_k = \imath(c_k - c_{-k}), k = 0, ..., M \tag{38}$$

(37) è detta anche **trasformata reale discreta di Fourier**.

Imponendo le condizioni di interpolazione nei nodi $t_j = jh$, con $h = \frac{2\pi}{m+1}, j = 0, ..., m$, troviamo

$$f_j = \sum_{k=-M}^{M} c_k e^{\imath kjh},\ j = 0, ..., m \tag{39}$$

che si riconduce alla soluzione di un sistema lineare

$$Tc = f$$

con $T = (e^{\imath kjh})_{k,j}$, c, f vettori di compomenti c_k e f_j rispettivamente.

La matrice è non singolare e quindi la soluzione è unica. Tale soluzione però può essere ottenua direttamente come segue

$$\sum_{j=0}^{m} f_j e^{-\imath \ell jh} = \sum_{j=0}^{m} f_j e^{-\imath \ell jh} \sum_{k=-M}^{M} c_k e^{\imath kjh},\ \ell = -M, ..., 0, ..., +M.$$

Poichè risulta

$$\sum_{j=0}^{m} e^{-\imath(k-\ell)jh} = \begin{cases} 0 \text{ se } k \neq \ell \\ (m+1) \text{ se } k \neq \ell \end{cases}$$

si ottiene

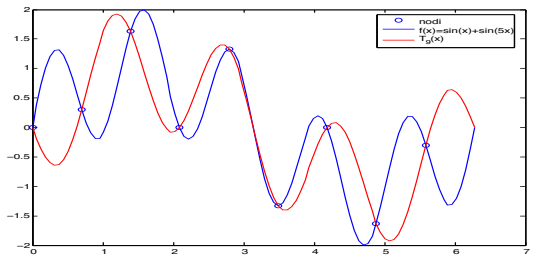
$$c_\ell = \frac{1}{m+1} \sum_{j=0}^{m} f_j e^{-\imath \ell jh}, \ell = -M, ..., 0, ..., +M.$$

Il calcolo della soluzione può quindi essere ottenuto con un prodotto matrice-vettore ed un costo dell'ordine di $(m+1)^2$ flops. Tale costo può essere significativamente ridotto a $(m+1)log_2(m+1)$ flops se si utilizza la **trasformata rapida di Fourier** o **FFT**.

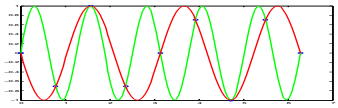
In Matlab si ha a disposizione la funzione fft

Anche $f = T^{-1}c$ (**trasformata discreta di Fourier inversa**) può essere calcolato in maniera efficiente con la versione rapida, che in Matlab è implementata da $ifft$.

L'interpolazione trigonometrica può essere poco accurata in particolari situazioni. Consideriamo la funzione $f(t) = sin(t) + sin(5t)$ e costruiamo il polinomio trigonometrico interpolante con $m = 8$.



I risultati ottenuti trovano spiegazione nel fatto che $sin(5t)$ e $-sin(3t)$ assumono gli stessi valori nei punti del campionamento $t_j = jh, j = 0, ...8, h = \frac{2*pi}{9}$ e quindi il polinomio trigonometrico ottenuto approssima la funzione $sin(t) - sin(3t)$.



Solo aumentando il numero di nodi sarà possibile approssimare correttamente la componente $sin(5t)$ di frequenza più alta della funzione, altrimenti indistinguibile da quella di frequenza più bassa $-sin(3t)$

Questo fenomeno prende il nome di aliasing: finchè la discretizzazione non è sufficientemente fine da distinguere le frequenze più elevate dalle più basse l'approssimazione sarà innacurata.

8 MATLAB

8.1 Interpolazione

Si consideri il polinomio di grado n

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

Tale polinomio viene rappresentato in MATLAB con il vettore

```
p=[p(1),p(2),...,p(n+1)]
```

di elementi $p(i) = a_{n-i+1}$ per $i = 1, 2, \ldots, n+1$. Assegnati $n+1$ punti distinti (x_i, y_i), $i = 1, 2, \ldots, n+1$, l'istruzione *polyfit* restituisce il polinomio p di grado n che interpola i punti dati. La sintassi è

```
p=polyfit(x,y,n)
```

dove x e y sono rispettivamente i vettori di elementi x_i e y_i. Tale polinomio è ottenuto risolvendo il sistema di Vandermonde (con la fattorizzazione LU), ovvero il vettore p ha come elementi i coefficienti del polinomio interpolante secondo la base dei monomi (vedi Esercizio 1). È possibile modificare l'M-file *polyfit.m* per cambiare la base di rappresentazione del polinomio (ad es. usando Lagrange o Newton).

Una volta calcolato, il polinomio p può essere valutato in un punto (o in un vettore di punti) z diverso dai nodi di interpolazione mediante l'istruzione *polyval* che utilizza l'algoritmo di Horner (vedi anche istruzioni MATLAB per le equazioni non lineari). La sintassi è

```
pz=polyval(p,z)
```

Un'alternativa a *polyfit* e *polyval* è l'istruzione *interp1*, la cui sintassi è

```
Y=interp1(x,y,X)
```

la quale restituisce il vettore di valori Y del polinomio interpolante i punti identificati dai vettori x e y valutato nel vettore di punti X. La sintassi alternativa

```
Y=interp1(x,y,X,'method')
```

permette di cambiare il tipo di interpolazione mediante la stringa *'method'*. Tra gli altri si possono utilizzare:

- *'nearest'*: costante a tratti
- *'linear'*: polinomiale (default)
- *'spline'*: spline cubiche
- *'cubic'*: interpolazione di Hermite con polinomi cubici a tratti.

Esistono istruzioni simili per l'interpolazione su domini di dimensione 2 e 3 (*interp2* e *interp3*).

Per l'interpolazione mediante spline cubiche esiste in MATLAB una *toolbox* dedicata. In particolare, il polinomio cubico a tratti *pp* che interpola i dati assegnati con i vettori x e y si ottiene con l'istruzione *spline*. La sintassi è

```
pp=spline(x,y)
```

(Si osserva che l'algoritmo utilizzato costruisce spline cubiche che non sono nè naturali, nè periodiche, nè tantomeno vincolate, ma soddisfano alle condizioni "not-a-knot" che sono particolari condizioni sulla derivata terza sul primo e ultimo sottointervallo della suddivisione). Successivamente, tale polinomio può essere valutato in un vettore di punti X mediante l'istruzione *ppval* con la seguente sintassi:

```
Y=ppval(pp,X)
```

o alternativamente, sempre con l'istruzione spline:

```
Y=spline(x,y,X)
```

Per quanto riguarda i nodi di interpolazione, nel caso siano equidistanti allora possono essere direttamente costruiti mediante l'istruzione *linspace*. La sintassi

```
x=linspace(a,b,n)
```

suddivide infatti l'intervallo $[a, b] \subset \mathbb{R}$ con n punti equidistanti raccolti nel vettore x dati da

$$x_i = a + i\frac{b-a}{n-1}, \quad i = 0, \ldots, n-1.$$

L'istruzione corrisponde banalmente a costruire il vettore

```
x=a:(b-a)/(n-1):b
```

Se n non viene specificato, allora di default è $n = 100$.

Si riporta infine, per completezza, l'istruzione *polyder* per il calcolo del polinomio *dp* derivata del polinomio *p*. La sintassi è

```
k=polyder(p)
```

9 Introduzione alla miglior approssimazione: minimi quadrati

Si consideri il sistema lineare sovradeterminato $Ax \approx b$ con $A \in \mathbb{R}^{m\times n}$, $m \gg n$, $b \in \mathbb{R}^m$ dato e $x \in \mathbb{R}^n$ incognito. In MATLAB, il sistema può essere risolto nel senso dei minimi quadrati semplicemente utilizzando l'istruzione

```
x=A\b
```

Se A ha rango pieno (i.e. $\text{rank}(A) = n$), allora dietro la precedente istruzione si cela la fattorizzazione $AP = QR$ (fattorizzazione QR con pivot di colonna) e la risoluzione di un sistema triangolare superiore (sostituzione all'indietro). Se invece $\text{rank}(A) = k < n$, allora la soluzione è ottenuta mediante

```
x=pinv(A)*b
```

dove *pinv* fornisce la pseudoinversa di A, ovvero $A^+ = (A^TA)^{-1}A^T$.

La fattorizzazione QR di A si ottiene esplicitamente con l'istruzione *qr*. La sintassi standard è

```
[Q,R]=qr(A)
```

e fornisce la matrice di trasformazione ortogonale $Q \in \mathbb{R}^{m\times m}$ e il fattore triangolare superiore $R \in \mathbb{R}^{m\times n}$. La sintassi

```
[Q,R]=qr(A,0)
```

realizza la fattorizzazione QR "economy size", ovvero calcola solo le prime n colonne della matrice di trasformazione ortogonale e restituisce $R \in \mathbb{R}^{n\times n}$. La sintassi

```
[Q,R,P]=qr(A)
```

realizza la fattorizzazione $AP = QR$ con pivot di colonna. Infine, le sintassi

```
R=qr(A)
```

e

```
R=qr(A,0)
```

restituiscono solo il fattore triangolare superiore R, rispettivamente con la fattorizzazione completa o "economy size".

Nota la fattorizzazione $A = QR$, il sistema può essere risolto con le seguenti istruzioni (vedi Esercizio 4):

```
>>c=Q'*b;
>>x=R(1:n,:)\c(1:n);
>>r=norm(c(n+1:m),2);
```

dove c è il vettore dei termini noti trasformato, x la soluzione ed r il residuo.

Il problema dell'approssimazione polinomiale nel senso dei minimi quadrati viene risolto in MATLAB con l'istruzione *polyfit* già vista per l'interpolazione. Assegnati m punti distinti (x_i, y_i), $i = 1, \ldots, m$, la sintassi

```
p=polyfit(x,y,n)
```

con $n < m - 1$ restituisce il polinomio p di grado n che soddisfa $p(x_i) = y_i$, $i = 1, \ldots, m$, nel senso dei minimi quadrati. x e y sono rispettivamente i vettori di elementi x_i e y_i. La costruzione del sistema lineare (vedi Esercizio 5) si basa sulla formazione della matrice di Vandermonde come per l'interpolazione. La risoluzione di tale sistema è ottenuta con l'operatore '\' come descritto sopra.

NOTA. Per tutti i comandi precedenti si consiglia di consultare l'*help* e la documentazione (*doc*) in linea di MATLAB.

10 Esercizi svolti

10.1 Interpolazione

Esercizio 1 *Si determini il polinomio di secondo grado $p \in \Pi_2$ che interpola i punti*

$$(x_0, y_0) = (-2, -27), \quad (x_1, y_1) = (0, -1), \quad (x_2, y_2) = (1, 0).$$

usando l'interpolazione nella base dei monomi, nella base di Lagrange e nella base di Newton.

Soluzione. Innanzitutto si osserva che, essendo i nodi di interpolazione x_i, $i = 0, 1, 2$, distinti, il polinomio interpolante esiste ed è unico. Esso può comunque essere rappresentato (e valutato) in modi differenti a seconda della scelta della base in Π_2. Si analizzano di seguito le basi dei monomi, di Lagrange e di Newton.

BASE DEI MONOMI. La rappresentazione del polinomio nella base dei monomi è

$$p_m(x) = M_0 m_0(x) + M_1 m_1(x) + M_2 m_2(x).$$

dove

$$m_j(x) = x^j, \quad j = 0, 1, 2,$$

ovvero (Figura 1)

$$\begin{aligned} m_0(x) &= 1 \\ m_1(x) &= x \\ m_2(x) &= x^2 \end{aligned}$$

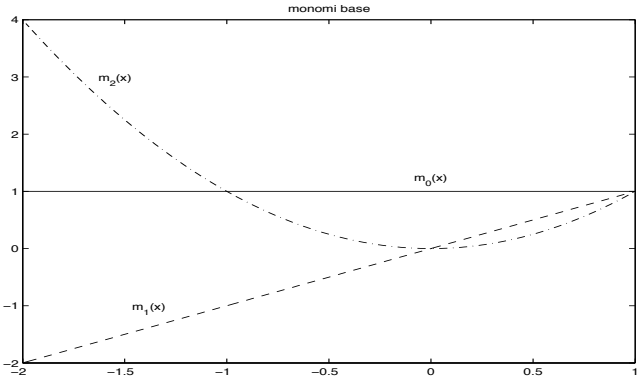


Figura 1: Base dei monomi.

I coefficienti della rappresentazione M_j, $j = 0, 1, 2$, si determinano con le condizioni di interpolazione

$$y_i = p_m(x_i), \quad i = 0, 1, 2.$$

Si ottiene così il sistema lineare

$$VM = y$$

dove

$$V = [m_j(x_i)]_{i,j=0,1,2} = \begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

è la matrice di Vandermonde calcolata rispetto ai nodi di interpolazione $x = (-2, 0, 1)^T$, $M = (M_0, M_1, M_2)^T$ è il vettore dei coefficienti di p_m e $y = (-27, -1, 0)^T$ è il vettore dei valori assunti da p_m nei nodi di interpolazione. Poichè questi ultimi sono distinti, V è non singolare (i.e. $\det V \neq 0$), per cui il sistema ammette l'unica soluzione (calcolabile ad esempio con la fattorizzazione LU, costo totale $\mathcal{O}(n^3/3)$ moltiplicazioni e $\mathcal{O}(n^3/3)$ addizioni)

$$M = (-1, 5, -4)^T$$

e risulta dunque

$$p_m(x) = -1 + 5x - 4x^2.$$

Qualora si voglia valutare p_m in un punto z diverso dai nodi di interpolazione, il metodo più efficiente (n moltiplicazioni ed n addizioni) è quello di Horner, che per il caso generale di un polinomio di grado n (ovvero per $n + 1$ nodi di interpolazione) corrisponde a

$$p_m(z) = M_0 + z(M_1 + z(M_2 + z(\cdots(M_{n-1} + zM_n)\cdots))).$$

Una possibile implementazione MATLAB dell'algoritmo è la seguente:

```
pmz=M(n+1);
for i=n-1:-1:0
    pmz=pmz.*z+M(i+1);
end
```

(NB: *in MATLAB i vettori e le matrici non ammettono l'indice zero*).

La matrice di Vandermonde è in generale mal condizionata per n grande. La situazione può essere migliorata con lo *scaling* dei nodi di interpolazione, ad esempio scegliendo come base

$$m_j(x) = \left(\frac{x - c}{d}\right)^j, \quad j = 0, 1, \ldots, n,$$

dove

$$c = \frac{x_0 + x_n}{2}$$

è il punto medio dell'intervallo e

$$d = \frac{x_n - x_0}{2}$$

la sua semiampiezza. Tale operazione può evitare anche eventuali *overflow* e *underflow*.

BASE DI LAGRANGE. La rappresentazione nella base di Lagrange è

$$p_l(x) = \lambda_0 l_0(x) + \lambda_1 l_1(x) + \lambda_2 l_2(x)$$

dove

$$l_j(x) = \prod_{i=0, i\neq j}^{2} \frac{x - x_i}{x_j - x_i}, \quad j = 0, 1, 2,$$

ovvero (Figura 2)

$$\begin{aligned}
l_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 0)(x - 1)}{(-2)(-3)} = \frac{x^2 - x}{6} \\
l_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_{10} - x_2)} = \frac{(x + 2)(x - 1)}{2(-1)} = -\frac{x^2 + x - 2}{2} \\
l_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x + 2)(x - 0)}{3 \cdot 1} = \frac{x^2 + 2x}{3}.
\end{aligned}$$

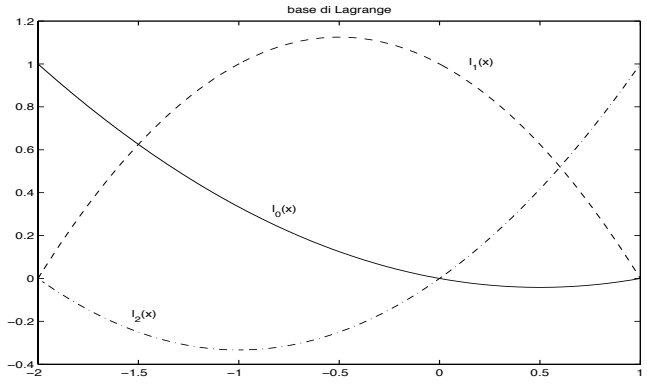


Figura 2: Base di Lagrange.

I coefficienti della rappresentazione λ_j, $j = 0, 1, 2$, si determinano con le condizioni di interpolazione

$$y_i = p_l(x_i), \quad i = 0, 1, 2.$$

Si ottiene così il sistema lineare

$$L\lambda = y$$

dove L è la matrice identica (i.e. $L = I$) e di conseguenza il vettore dei coefficienti risulta $\lambda = y$. Per la base di Lagrange si ha infatti

$$l_j(x_i) = \begin{cases} 1 \text{ per } i = j \\ 0 \text{ per } i \neq j \end{cases}$$

Segue

$$p_l(x) = -27\frac{x^2 - x}{6} + \frac{x^2 + x - 2}{2} = -1 + 5x - 4x^2 = p_m(x).$$

Per valutare p_l in un punto z diverso dai nodi di interpolazione è necessario valutare ciascun polinomio della base in tale punto. Si potrebbero calcolare i coefficienti di p_l nella base dei monomi e poi applicare il metodo di Horner, ma ciò comporterebbe un costo totale di moltiplicazioni e addizioni superiore a $6n^2$ solo per la determinazione dei coefficienti. Un'alternativa più efficiente consiste nel considerare la seguente formulazione:

$$p_l(z) = \pi_n(z) \sum_{j=0}^{n} \frac{v_j}{z - x_j}$$

dove

$$\pi_n(z) = \prod_{i=0}^{n} (z - x_i)$$

e

$$v_j = \frac{y_j}{\prod\limits_{i=0, i\neq j}^{n} (x_j - x_i)}, \quad j = 0, 1, \ldots, n.$$

In tal caso il costo totale scende a $\mathcal{O}(n^2)$ moltiplicazioni e $\mathcal{O}(n^2/2)$ addizioni. Se inoltre si cambia il punto z, la nuova valutazione comporta solo l'aggiunta di $\mathcal{O}(2n)$ moltiplicazioni e $\mathcal{O}(2n)$ addizioni dato che i v_j dipendono esclusivamente dai nodi. Infine, se si aggiunge un nodo di interpolazione (x_{n+1}, y_{n+1}), per valutare il nuovo polinomio nello stesso punto z sono richieste solo $2n$ moltiplicazioni e $2n$ addizioni aggiuntive dato che si possono utilizzare le quantità $\pi_n(x)$ e $\frac{v_j}{z - x_j}$ già computate in precedenza (vedi Tabella 2). La risoluzione "gratuita" del sistema lineare è quindi bilanciata dalla valutazione del polinomio in un punto, più costosa rispetto al metodo di Horner. In compenso il condizionamento è perfetto dato che la matrice del sistema è l'identità.

BASE DI NEWTON. La rappresentazione nella base di Newton è

$$p_n(x) = \nu_0\pi_0(x) + \nu_1\pi_1(x) + \nu_2\pi_2(x)$$

dove

$$\pi_j(x) = \begin{cases} \prod\limits_{i=0}^{j-1} (x - x_i) & \text{per } j = 1, 2 \\ 1 & \text{per } j = 0 \end{cases}$$

ovvero (Figura 3)

$$\pi_0(x) = 1$$
$$\pi_1(x) = x - x_0 = x + 2$$
$$\pi_2(x) = (x - x_0)(x - x_1) = (x + 2)x.$$

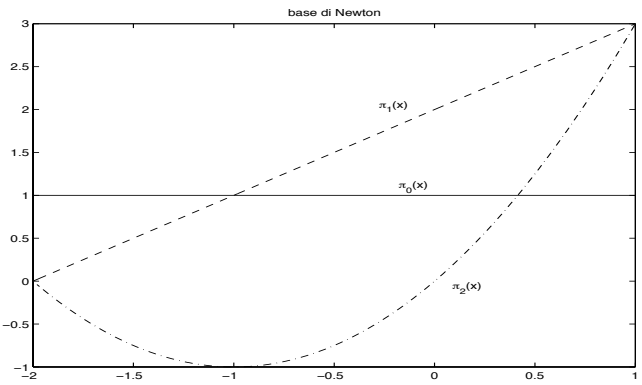


Figura 3: Base di Newton.

I coefficienti della rappresentazione ν_j, $j = 0, 1, 2$, si determinano con le condizioni di interpolazione

$$y_i = p_n(x_i), \quad i = 0, 1, 2.$$

Si ottiene così il sistema lineare

$$N\nu = y$$

dove

$$N = [\pi_j(x_i)]_{i,j=0,1,2} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & x_1 - x_0 & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{pmatrix}$$

è sempre triangolare inferiore. La costruzione di N costa $\mathcal{O}(n^2/2)$ moltiplicazioni e $\mathcal{O}(n^2/2)$ addizioni, tanto quanto la risoluzione per sostituzione in avanti del sistema: il costo totale è dunque $\mathcal{O}(n^2)$ moltiplicazioni e $\mathcal{O}(n^2)$ addizioni. Si ottiene così il vettore dei coefficienti

$$\nu = (-27, 13, -4)^T$$

e combinando con i polinomi base risulta

$$p_n(x) = -27 + 13(x + 2) - 4(x + 2)x = -1 + 5x - 4x^2 = p_m(x).$$

k	i	F
1	1	$F[x_0,x_1] = \frac{F[x_1]-F[x_0]}{x_1-x_0}$
	2	$F[x_0,x_2] = \frac{F[x_2]-F[x_0]}{x_2-x_0}$
	$\vdots$	$\vdots$
	n	$F[x_0,x_n] = \frac{F[x_n]-F[x_0]}{x_n-x_0}$
2	2	$F[x_0,x_1,x_2] = \frac{F[x_0,x_2]-F[x_0,x_1]}{x_2-x_1}$
	3	$F[x_0,x_1,x_3] = \frac{F[x_0,x_3]-F[x_0,x_1]}{x_3-x_1}$
	$\vdots$	$\vdots$
	n	$F[x_0,x_1,x_n] = \frac{F[x_0,x_n]-F[x_0,x_1]}{x_n-x_1}$
$\cdots$		
n	n	$F[x_0,\ldots,x_n] = \frac{F[x_0,\ldots,x_{n-2},x_n]-F[x_0,x_1,\ldots,x_{n-1}]}{x_n-x_{n-1}}$

Tabella 1: Costruzione delle differenze divise.

Per valutare p_n in un punto z si può applicare la seguente variante del metodo di Horner:

$$p_n(z) = \nu_0 + (z-x_0)(\nu_1 + (z-x_1)(\nu_2 + (z-x_2)(\cdots(\nu_{n-1} + (z-x_{n-1})\nu_n)\cdots)))$$

che costa n moltiplicazioni e $2n$ addizioni.

La base di Newton offre il miglior compromesso tra il costo della determinazione del polinomio interpolante e la valutazione dello stesso in un punto. Il costo della determinazione dei coefficienti può essere ulteriormente ridotto: i coefficienti del polinomio di Newton si possono infatti ottenere alternativamente utilizzando le **differenze divise**, dimezzando il numero delle moltiplicazioni ed evitando anche situazioni di *overflow* e *underflow*. In generale, dati $n+1$ punti (x_i, y_i), $i = 0, 1, \ldots, n$, la differenza divisa di ordine k è definita ricorsivamente come

$$f[x_i] = y_i$$

per $i = 0, \ldots, n$ e

$$f[x_0, x_1, \ldots, x_{k-1}, x_i] = \frac{f[x_0, x_1, \ldots, x_{k-2}, x_i] - f[x_0, x_1, \ldots, x_{k-1}]}{x_i - x_{k-1}}$$

per $i = k, \ldots, n$ e $k = 1, \ldots, n$. Dopo aver impostato dunque le differenze divise di ordine 1 (impostato e non calcolato dato che corrispondono ai valori y_i, $i = 0, 1, \ldots, n$), le differenze di ordine superiore vengono determinate secondo la Tabella 1. Si ottiene così lo schema

$$\begin{array}{c|ccccc} x_0 & f[x_0] & & & & \\ x_1 & f[x_1] & f[x_0,x_1] & & & \\ x_2 & f[x_2] & f[x_0,x_2] & f[x_0,x_1,x_2] & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ x_n & f[x_n] & f[x_0,x_n] & f[x_0,x_1,x_n] & \cdots & f[x_0,x_1,\ldots,x_n] \end{array}$$

i cui elementi, considerati come elementi di una matrice triangolare inferiore F, possono essere calcolati con il seguente algoritmo:

$$f_{i0} = y_i \text{ per } i = 0, \ldots, n$$
$$f_{ij} = \frac{f_{ij-1} - f_{j-1j-1}}{x_i - x_{j-1}} \text{ per } j = 1, \ldots, n, \quad i = j, \ldots, n$$

che richiede un totale di $\mathcal{O}(n^2/2)$ moltiplicazioni e $\mathcal{O}(n^2)$ addizioni. Sfruttando la notazione compatta *colon* di MATLAB, il precedente algoritmo può essere implementato in modo efficiente con la seguente pseudocodifica:

```
F=y;
for j=1:n
    F(j+1:n+1)=(F(j+1:n+1)-F(j))./(x(j+1:n+1)-x(j));
end
```

Tale implementazione trasforma il vettore y dei valori nei nodi di interpolazione x nel vettore delle differenze divise costituito dalla diagonale principale di F

$$y \mapsto (f[x_0], f[x_0, x_1], f[x_0, x_1, x_2], \ldots, f[x_0, x_1, \ldots, x_n])^T.$$

Si può dimostrare che il polinomio nella base di Newton è rappresentabile come

$$p_n(x) = f[x_0]\pi_0(x) + f[x_0, x_1]\pi_1(x) + \cdots + f[x_0, x_1, \ldots, x_n]\pi_n(x)$$

ovvero

$$\nu_j = f[x_0, x_1, \ldots, x_j], \quad j = 0, 1, \ldots, n.$$

La valutazione in un punto z si ottiene sempre con il metodo di Horner. Le differenze divise non cambiano per permutazioni dell'ordine dei punti, ma tale ordinamento influisce sul condizionamento nel calcolo dei coefficienti di p_n e non è detto che il naturale ordinamento sx-dx sia il migliore.

Uno dei vantaggi nell'utilizzare le differenze divise è che l'aggiunta di un punto di interpolazione (x_{n+1}, y_{n+1}) richiede solo il calcolo di un'ulteriore riga della matrice F (la $(n+1)$-esima), al costo di n moltiplicazioni e $2n$ addizioni mentre la sua valutazione in z richiede solo n moltiplicazioni aggiuntive per il calcolo di $\pi_{n+1}(z)$:

$$p_{n+1}(z) = p_n(z) + f[x_0, x_1, \ldots, x_{n+1}]\pi_{n+1}(z).$$

La valutazione del nuovo polinomio nello stesso punto z comporta dunque un totale di $2n$ moltiplicazioni e $2n$ addizioni aggiuntive come per il polinomio di Lagrange (vedi Tabella 2). L'algoritmo proposto in precedenza non è efficiente se si prevede l'aggiunta di nodi dato che in tal caso bisogna conservare l'intera tabella delle differenze divise (e non solo la diagonale) per il calcolo di ulteriori righe. Infine, il calcolo del polinomio di Newton mediante le differenze divise risulta in certi casi più stabile del calcolo effettuato con il polinomio di Lagrange (e ancor più del caso dei monomi).

Con i dati assegnati la tabella delle differenze divise risulta

	polinomio di Lagrange	polinomio di Newton
calcolo in un punto	$n^2M + \frac{n^2}{2}A$	$\frac{n^2}{2}M + n^2A$
aggiunta di un nodo	$2nM + 2nA$	$2nM + 2nA$

Tabella 2: Complessità computazionale per l'interpolazione di Lagrange e di Newton: M moltiplicazioni e A addizioni (si considerano solo i termini di ordine maggiore).

$$\begin{array}{r|lll} -2 & f[x_0] = -27 & & \\ 0 & f[x_1] = -1 & f[x_0, x_1] = 13 & \\ 1 & f[x_2] = 0 & f[x_0, x_2] = 9 & f[x_0, x_1, x_2] = -4 \end{array}$$

per cui la diagonale $(-27, 13, -4)^T$ contiene effettivamente i coefficienti del polinomio nella base di Newton.

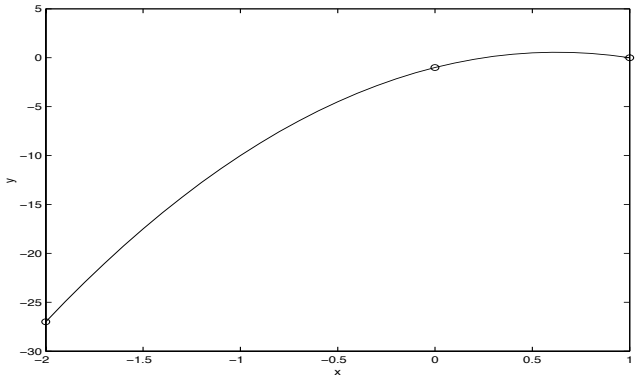


Figura 4: Polinomio interpolante.

Il polinomio interpolante $p = p_m = p_l = p_n$ è disegnato in Figura 4: dato il basso grado, le tre rappresentazioni graficamente coincidono anche se vi sono delle discrepanze di poco superiori alla precisione macchina dovute ad instabilità algoritmiche. Utilizzando MATLAB si ottiene infatti:

```
>>[z,abs(pnz-plz),abs(pnz-pmz)]
ans=
 -2.0000e+000  7.1054e-015            0
 -1.9000e+000  3.5527e-015            0
 -1.8000e+000  7.1054e-015            0
 -1.7000e+000  7.1054e-015  3.5527e-015
 -1.6000e+000  3.5527e-015            0
 -1.5000e+000  3.5527e-015            0
 -1.4000e+000  1.7764e-015            0
 -1.3000e+000             0           0
```

```
-1.2000e+000  3.5527e-015  1.7764e-015
-1.1000e+000            0  1.7764e-015
-1.0000e+000  1.7764e-015            0
-9.0000e-001            0            0
-8.0000e-001            0  8.8818e-016
-7.0000e-001  1.7764e-015  2.6645e-015
-6.0000e-001  1.7764e-015  2.6645e-015
-5.0000e-001            0            0
-4.0000e-001  8.8818e-016  4.4409e-016
-3.0000e-001  2.6645e-015  2.6645e-015
-2.0000e-001  8.8818e-016  8.8818e-016
-1.0000e-001  1.5543e-015  1.3323e-015
           0            0            0
 1.0000e-001  8.8818e-016  9.9920e-016
 2.0000e-001            0            0
 3.0000e-001  7.7716e-016  6.6613e-016
 4.0000e-001  2.2204e-016  4.4409e-016
 5.0000e-001  3.3307e-016            0
 6.0000e-001  7.7716e-016  1.3323e-015
 7.0000e-001            0  8.8818e-016
 8.0000e-001  2.4425e-015  1.3323e-015
 9.0000e-001  2.7756e-015  1.5543e-015
 1.0000e+000  1.2768e-015            0
```

■

Esercizio 2 *Si interpoli la funzione $f(x) = log(x)$ nell'intervallo $[a, b] = [2, 3]$ su $n + 1$ nodi equidistanti analizzando il comportamento dell'errore relativo in un dato punto z al crescere di n e su tutto l'intervallo a parità di n.*

Soluzione. Innanzitutto si generano gli $n+1$ nodi equidistanti. In un generico intervallo $[a, b] \in \mathbb{R}$, questi sono dati da

$$x_i = a + i\frac{b-a}{n}, \quad i = 0, 1, \ldots, n.$$

Per l'analisi dell'errore relativo in z

$$\epsilon_{rel} = \frac{|log(z) - p(z)|}{|log(z)|}$$

si sono create delle *function* MATLAB che valutano il polinomio interpolante sia nella base di Lagrange che in quella di Newton. In particolare, i polinomi base di Lagrange vengono valutati nel punto z secondo due algoritmi differenti. Il primo si basa sulla costruzione dei polinomi base mediante l'istruzione *prod* che interpreta direttamente la definizione di $l_j(z)$:

```
%% Lagrange, algoritmo 1, dati x,n,z %%
```

```
xx=[x(1:j);x(j+2:n+1)];
ljz=prod(z-xx)/prod(x(j+1)-xx);
```

Il secondo, invece, si basa sull'uso di *polyfit* e *polyval*, ovvero della matrice di Vandermonde. Il corpo centrale è

```
%% Lagrange, algoritmo 2, dati x,n,z %%

I=eye(n+1);
lj=polyfit(x,I(:,j+1),n);
ljz=polyval(lj,z);
```

dove il j-esimo polinomio base di Lagrange l_j è costruito osservando che deve interpolare gli $n+1$ punti (x, e_j) dove e_j è il j-esimo vettore della base canonica ovvero la j-esima colonna della matrice identità. Tale polinomio è poi valutato in z mediante *polyval*.

Per quanto riguarda l'interpolazione di Newton, anche qui si sono utilizzati due algoritmi differenti. Il primo costruisce la diagonale delle differenze divise *dd* in forma compatta come spiegato nell'Esercizio 1:

```
%% Newton, algoritmo 1, dati x,y=log(x),n %%

F=y;
for j=1:n
    F(j+1:n+1)=(F(j+1:n+1)-F(j))./(x(j+1:n+1)-x(j));
end
```

Il secondo, invece, costruisce la diagonale delle differenze divise F risolvendo il sistema triangolare inferiore $N\nu = y$ mediante l'operatore \ (quindi per sostituzione in avanti), dato che il vettore dei coefficienti ν coincide con F per quanto visto nell'Esercizio 1.

```
%% Newton, algoritmo 2, dati x,y=log(x),n %%

N(:,1)=ones(n+1,1);
for j=1:n
    for i=1:n+1
        N(i,j+1)=prod(x(i)-x(1:j));
    end
end
F=N\y;
```

Si fa osservare che altre implementazioni (anche più efficienti) sono possibili.

Nelle Figure 5 e 6 si è calcolato l'errore relativo in $z = 2.008$ per $n = 1, 2, \ldots, 50$, rispettivamente con i due algoritmi di Lagrange e i due di Newton. Per Lagrange, l'algoritmo 1 fornisce un errore che inizialmente decresce e poi cresce: all'inizio prevale l'errore analitico mentre per n grande l'errore algoritmico è predominante. L'algortimo 2, invece, fornisce un'errore del 100% con

$n > 12$, conseguenza del mal condizionamento della matrice di Vandermonde. Per Newton l'andamento dei due algoritmi è pressochè identico confermando il compromesso tra l'errore analitico e quello algoritmico. Nelle figure 7 e 8 si

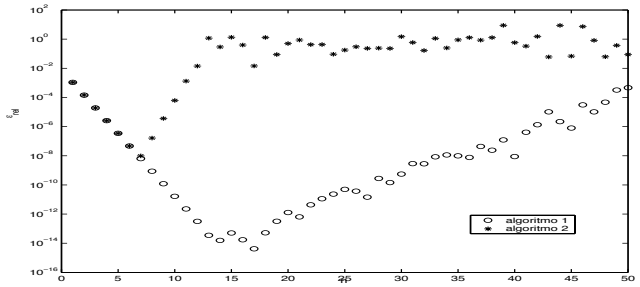


Figura 5: Interpolazione di Lagrange, $z = 2.008$.

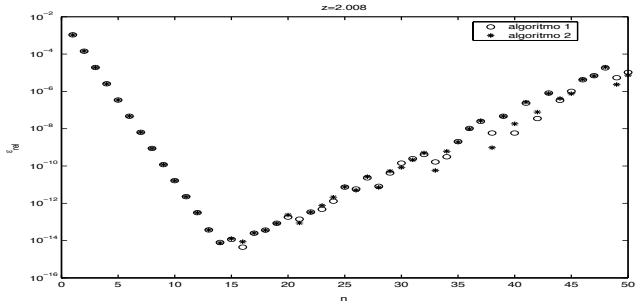


Figura 6: Interpolazione di Newton, $z = 2.008$.

riportano i risultati per $z = 2.52$: al centro dell'intervallo l'approssimazione è migliore e l'errore algoritmico è praticamente ininfluente. Nel caso dell'uso della matrice di Vandermonde, invece, il mal condizionamento mantiene un ruolo fondamentale continuando a fornire errori del 100%. Dunque la stabilità degli algoritmi usati gioca un ruolo tanto più determinante quanto più si è vicini agli estremi dell'intervallo considerato.

Infine, si sono testati i secondi algoritmi per ciascuna base valutando l'errore relativo sull'intero intervallo per $n = 29$. Si osserva in figura 10.1 come l'interpolazione di Newton fornisca risultati leggermente migliori, comportamento che si verifica in molti casi data la maggior stabilità delle differenze divise rispetto alla base di Lagrange.

Gli errori algoritmici sono essenzialmente dovuti al fenomeno della cancellazione sia nel caso di Lagrange che in quello delle differenze divise. ■

Esercizio 3 *Si analizzi l'interpolazione polinomiale su $n+1$ punti distinti nell'intervallo $[a, b] \subset \mathbb{R}$ delle funzioni $f(x) = |x|$ e $f(x) = \frac{1}{1+x^2}$ (funzione di Runge). In particolare si considerino i punti equidistanti e i punti di Chebyshev.*

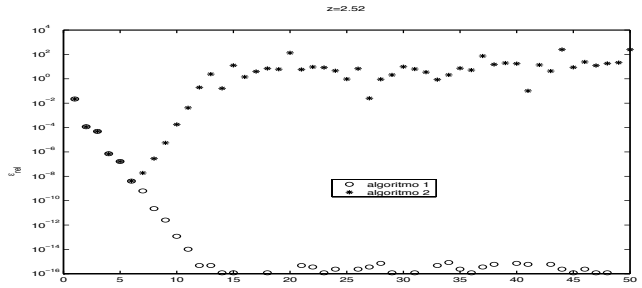


Figura 7: Interpolazione di Lagrange, $z = 2.52$.

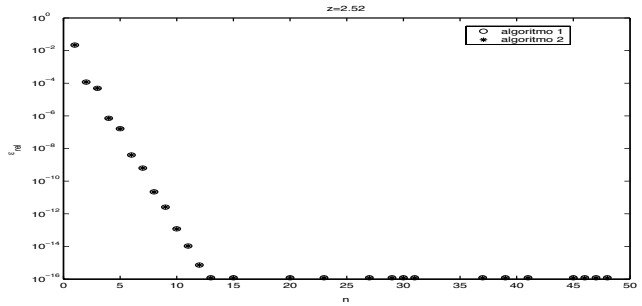


Figura 8: Interpolazione di Newton, $z = 2.52$.

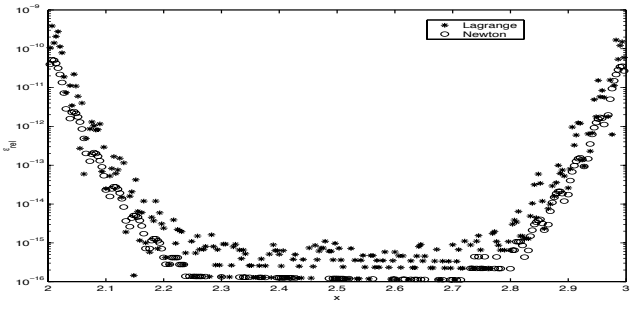


Figura 9: Interpolazione di Newton e di Lagrange sull'intero intervallo, $n = 29$.

Si valuti come varia l'errore dell'interpolazione al crescere di n e al variare del punto considerato in $[a, b]$.

Soluzione. Data una funzione $y = f(x)$ in un intervallo $[a, b] \subset \mathbb{R}$, il problema dell'interpolazione polinomiale consiste nel determinare il polinomio p_n di grado n che approssima f assumendone i valori $p_n(x_i) = y_i = f(x_i)$ in $n + 1$ punti $x_i \in [a, b]$, $i = 0, \ldots, n$, detti nodi di interpolazione.

Il problema dell'interpolazione polinomiale ammette sempre un'unica soluzione se gli $n+1$ punti x_i sono distinti. Considerato infatti il polinomio di grado n

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

il sistema lineare risultante dalle condizioni di interpolazione

$$\begin{cases} y_0 = p_n(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n \\ y_1 = p_n(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n \\ y_2 = p_n(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_n x_2^n \\ \cdots \\ y_n = p_n(x_n) = a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n \end{cases}$$

ovvero

$$Va = y$$

dove $V = [{x_i}^j]_{i,j=0}^n$, $a = (a_0, \ldots, a_n)^T$, $y = (y_0, \ldots, y_n)^T$, è quadrato ed essendo V di *Vandermonde*, allora è invertibile nella suddetta ipotesi. Segue che il vettore dei coefficienti $a = V^{-1}y$ esiste, è unico e caratterizza univocamente p_n.

Ragionando intuitivamente si sarebbe portati a pensare che il polinomio interpolante p_n approssima la funzione f tanto meglio al crescere del numero dei nodi utilizzati, i.e. al cresere di n. Al limite $p_n \to f$ per $n \to \infty$. Questo non è affatto vero in quanto **la convergenza del polinomio interpolante p_n alla funzione f dipende dalla scelta dei nodi x_i nonchè dalla regolarità della funzione f**. Per dare prova di quanto detto si riportano i risultati dell'interpolazione di $f(x) = |x|$ e di $f(x) = \frac{1}{1+x^2}$ (funzione di runge) nell'intervallo $[a, b] = [-5, 5]$, scegliendo due tipologie diverse per i nodi:

$$x_i = -a + i\frac{b-a}{n}, \quad i = 0, \ldots, n, \quad \text{nodi equidistanti}$$

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2i+1}{2n+2}\pi\right), \quad i = 0, \ldots, n, \quad \text{nodi di Chebyshev}$$

Il secondo caso corrisponde a scegliere come nodi gli zeri del polinomio ortogonale di Chebyshev di grado n. Naturalmete sono possibili altre scelte, le quali porteranno ad altri risultati.

Le Figure 10, 11 e 12 riguardano l'interpolazione di $f(x) = |x|$: aumentando n (5, 10, 20) si osserva come p_n approssimi f sempre peggio vicino agli estremi dell'intervallo per i nodi equidistanti, mentre l'approssimazione migliora su tutto l'intervallo per i nodi di Chebyshev. In Figura 13 si analizza invece l'errore massimo in norma-∞ per $n = 1, 2, \ldots, 50$: l'andamento conferma che la scelta

dei nodi di Chebyshev è migliore (l'aumento finale dell'errore per questi ultimi è dovuto all'instabilità numerica dell'algoritmo *polyval* e non all'errore analitico dell'approssimazione).

Le Figure 14, 15 e 17 riguardano l'interpolazione della funzione di runge $y = \frac{1}{1+x^2}$ e vi si riscontra il medesimo comportamento: l'interpolazione su nodi equidistanti non garantisce la convergenza per $n \to \infty$.

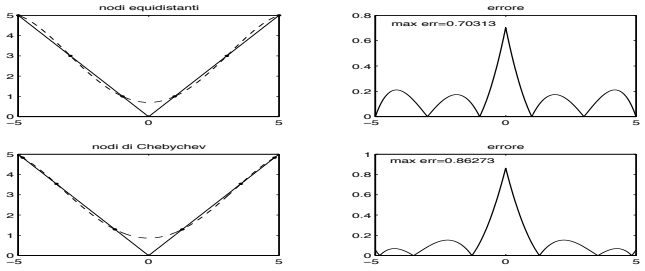


Figura 10: Interpolazione di $y = |x|$ per $n = 5$.

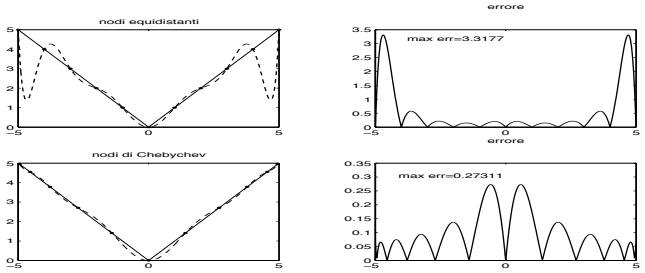


Figura 11: Interpolazione di $y = |x|$ per $n = 10$.

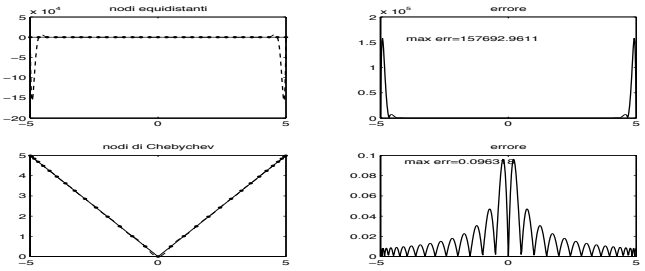


Figura 12: Interpolazione di $y = |x|$ per $n = 20$.

■

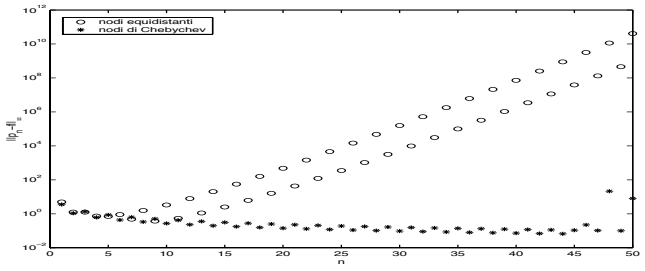


Figura 13: Errore massimo per $n = 1, 2, \ldots, 50$.

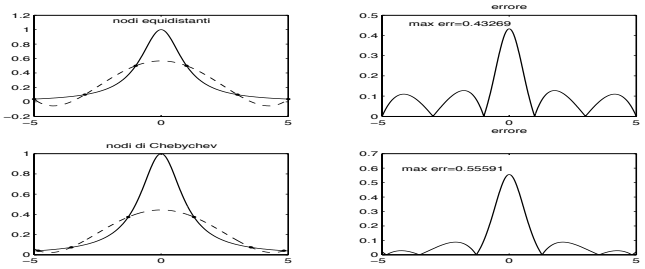


Figura 14: Interpolazione di $y = \frac{1}{1+x^2}$ per $n = 5$.

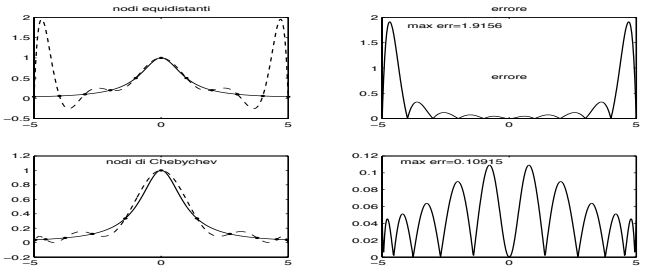


Figura 15: Interpolazione di $y = \frac{1}{1+x^2}$ per $n = 10$.

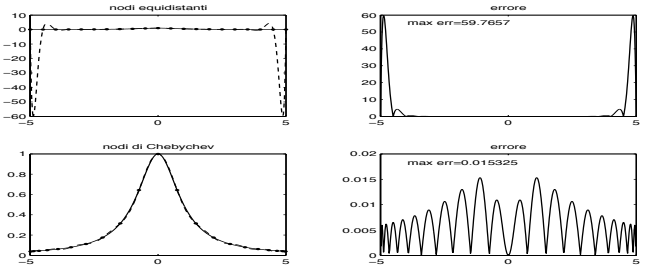


Figura 16: Interpolazione di $y = \frac{1}{1+x^2}$ per $n = 20$.

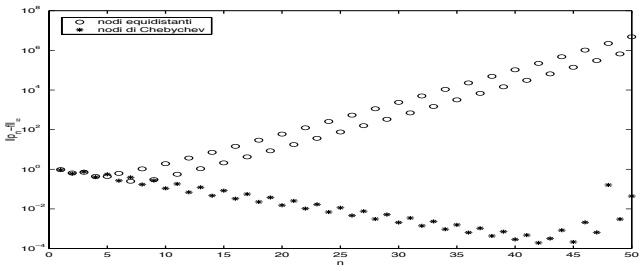


Figura 17: Errore massimo per $n = 1, 2, \ldots, 50$.

10.2 Minimi quadrati

Esercizio 4 *(vedi anche* $24/03/03 - \sharp 5$ *e* $16/09/03 - \sharp 6$*) Si vuole determinare la soluzione nel senso dei minimi quadrati del sistema lineare* $Ax \approx b$*, dove*

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \in \mathbb{R}^{m \times n} = \mathbb{R}^{4 \times 2}, \quad b = \begin{pmatrix} 1 \\ -4 \\ 3 \\ 2 \end{pmatrix} \in \mathbb{R}^m = \mathbb{R}^4.$$

Soluzione. La via più immediata per determinare la soluzione $x \in \mathbb{R}^n = \mathbb{R}^2$ che minimizza in norma-2 il residuo $r = b - Ax$ consiste nel costruire il sistema delle equazioni normali $A^T Ax = A^T b$. Se, infatti, A ha rango pieno ($rank(A) = n = 2$), allora $A^T A$ è definita positiva. Essendo poi $A^T A$ simmetrica, il sistema normale può essere efficientemente risolto mediante la fattorizzazione di Cholesky $A^T A = LL^T$ e la risoluzione per sostituzione dei due sistemi triangolari $Ly = A^T b$ (in avanti) e $L^T x = y$ (all'indietro). Si ottiene così

$$A^T A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 4 & 6 \\ 6 & 14 \end{pmatrix}$$

$$A^T b = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -4 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \end{pmatrix}$$

$$L = \begin{pmatrix} 2 & 0 \\ 3 & \sqrt{5} \end{pmatrix}$$

$$Ly = A^T b \Rightarrow \begin{pmatrix} 2 & 0 \\ 3 & \sqrt{5} \end{pmatrix} y = \begin{pmatrix} 2 \\ 8 \end{pmatrix} \Rightarrow y = \begin{pmatrix} 1 \\ \sqrt{5} \end{pmatrix}$$

$$L^T x = y \Rightarrow \begin{pmatrix} 2 & 3 \\ 0 & \sqrt{5} \end{pmatrix} x = \begin{pmatrix} 1 \\ \sqrt{5} \end{pmatrix} \Rightarrow x = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

In particolare si ottiene il residuo

$$r = b - Ax = \begin{pmatrix} 1 \\ -4 \\ 3 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ -4 \\ 2 \\ 0 \end{pmatrix} \Rightarrow \|r\|_2 = 2\sqrt{6}.$$

Il metodo illustrato è di immediata applicazione, ma la sua semplicità cela delle difficoltà numeriche che ne sconsigliano l'uso quando si opera in aritmetica di macchina. In primo luogo le perturbazioni sugli elementi di A possono dar luogo alla singolarità della matrice A^TA, in tal caso non ci sarebbe soluzione (si provi ad esempio con

$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{pmatrix}$$

per $0 < \epsilon < \sqrt{\epsilon_{macch}}$). In secondo luogo il sistema normale presenta un numero di condizionamento

$$cond(A^TA) = (cond(A))^2$$

per cui sistemi ben condizionati possono diventare mal condizionati.

Un'efficiente alternativa al metodo delle equazioni normali è l'utilizzo di trasformazioni ortogonali Q, cioè tali che $Q^TQ = I$, per ridurre il sistema iniziale ad uno più semplice conservando la norma-2 della soluzione. L'algoritmo di fattorizzazione QR, per esempio, trasforma il sistema iniziale in uno triangolare superiore con matrice $R \in \mathbb{R}^{n\times n}$ che può essere quindi risolto per sostituzione all'indietro. Infatti, tramite la fattorizzazione

$$A = Q \begin{pmatrix} R \\ \emptyset \end{pmatrix}$$

si ha che

$$\|r\|_2^2 = \left\| b - Q \begin{pmatrix} R \\ \emptyset \end{pmatrix} x \right\|_2^2 = \left\| Q^Tb - \begin{pmatrix} R \\ \emptyset \end{pmatrix} x \right\|_2^2 = \|c_1 - Rx\|_2^2 + \|c_2\|_2^2$$

dove

$$Q^Tb = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

è partizionato in modo che $c_1 \in \mathbb{R}^n$. Il residuo sarà dunque minimo quando x è soluzione del sistema $Rx = c_1$ e risulta $\|r\|_2 = \|c_2\|_2$.

La fattorizzazione QR si basa sull'eliminazione successiva (colonna per colonna) degli elementi di A sottostanti la diagonale principale. Questo si ottiene, similmente alla fattorizzazione LU, mediante matrici di trasformazione elementari che sono ortogonali. Il metodo più diffuso utilizza le trasformazioni di *Householder*

$$H = I_m - 2\frac{vv^T}{v^Tv}$$

che riflettono il vettore a cui sono applicate rispetto all'iperpiano $span(v)^\perp = \{z \in \mathbb{R}^m \,|\, v^T z = 0\}$ (vedi Figura 18). Si consideri come esempio il vettore $a \in \mathbb{R}^m$ partizionato secondo

$$a = \begin{pmatrix} \overline{a}_1 \\ \overline{a}_2 \end{pmatrix}$$

dove $\overline{a}_1 \in \mathbb{R}^{k-1}$ e $\overline{a}_2 \in \mathbb{R}^{m-k+1}$, $1 \leq k < m$. Se si prende

$$v = \begin{pmatrix} 0 \\ \overline{a}_2 \end{pmatrix} - \alpha e_k$$

dove $\alpha = -\text{sign}(a_k)\|\overline{a}_2\|_2$ ed e_k è il k-esimo vettore della base canonica, allora il risultato della trasforamzione di *Householder* applicata ad a sarà l'eliminazione delle ultime $m-k$ componenti di a conservando comunque la sua norma-2, ovvero $\|Ha\|_2 = \|a\|_2$. Si osservi che l'applicazione della trasformazione non richiede il calcolo esplicito di H in quanto basta costruire il vettore v:

$$Ha = \left(I_m - 2\frac{vv^T}{v^T v}\right) a = a - 2\frac{v^T a}{v^T v} v$$

dove $\frac{v^T a}{v^T v}$ è uno scalare.

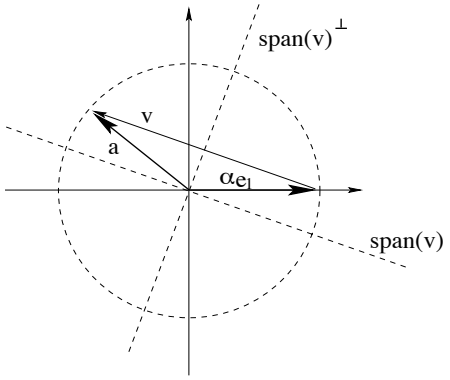


Figura 18: Trasformazione di Householder del vettore a.

Si applichi dunque la fattorizzazione QR al sistema $Ax \approx b$. Il primo passo è la costruzione del vettore di *Householder* v_k per eliminare gli elementi della prima colonna di A sotto la diagonale principale, quindi per $k = 1$. Si ha

$$\alpha_1 = -\text{sign}(a_{11})\|\overline{a}_{1,2}\|_2 = -2, \quad v_1 = \overline{a}_1 - \alpha_1 e_1 = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

per cui la prima colonna diventa

$$\overline{a}'_1 = \overline{a}_1 - 2\frac{v_1^T \overline{a}_1}{v_1^T v_1} v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - 2 \cdot \frac{6}{12} \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

mentre per la seconda colonna si ottiene

$$\overline{a}'_2 = \overline{a}_2 - 2\frac{v_1^T \overline{a}_2}{v_1^T v_1} v_1 = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} - 2 \cdot \frac{6}{12} \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ 0 \\ 1 \\ 2 \end{pmatrix}.$$

Il secondo passo consiste nell'eliminare gli elementi della seconda colonna aggiornata di A sotto la diagonale principale, quindi per $k = 2$. Si ha

$$\alpha_2 = -\text{sign}(a'_{22}) \|\overline{a}'_{2,2}\|_2 = -\sqrt{5}, \quad v_2 = \begin{pmatrix} 0 \\ \overline{a}'_{2,2} \end{pmatrix} - \alpha_2 e_2 = \begin{pmatrix} 0 \\ \sqrt{5} \\ 1 \\ 2 \end{pmatrix}$$

per cui la seconda colonna diventa

$$\overline{a}''_2 = \overline{a}'_2 - 2\frac{v_2^T \overline{a}'_2}{v_2^T v_2} v_2 = \begin{pmatrix} -3 \\ 0 \\ 1 \\ 2 \end{pmatrix} - 2 \cdot \frac{5}{10} \begin{pmatrix} 0 \\ \sqrt{5} \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} -3 \\ -\sqrt{5} \\ 0 \\ 0 \end{pmatrix}$$

mentre la prima colonna rimane inalterata. La matrice A è dunque ridotta ad

$$A = (\overline{a}'_1, \overline{a}''_2) = \begin{pmatrix} -2 & -3 \\ 0 & -\sqrt{5} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} R \\ \emptyset \end{pmatrix}.$$

Per determinare la soluzione x bisogna aggiornare il vettore b dei termini noti, ottenendo le partizioni c_1 e c_2 mediante le stesse trasformazioni v_1 e v_2 precedentemente calcolate:

$$b' = \overline{b} - 2\frac{v_1^T b}{v_1^T v_1} v_1 = \begin{pmatrix} 1 \\ -4 \\ 3 \\ 2 \end{pmatrix} - 2 \cdot \frac{4}{12} \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -\frac{14}{3} \\ \frac{7}{3} \\ \frac{4}{3} \end{pmatrix},$$

$$c = b' - 2\frac{v_2^T b'}{v_2^T v_2} v_2 = \begin{pmatrix} -1 \\ -\frac{14}{3} \\ \frac{7}{3} \\ \frac{4}{3} \end{pmatrix} - 2 \cdot \frac{15 - 14\sqrt{5}}{30} \begin{pmatrix} 0 \\ \sqrt{5} \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} -1 \\ -\sqrt{5} \\ \frac{2}{3}\left(2 + \frac{7}{\sqrt{5}}\right) \\ \frac{2}{3}\left(-1 + \frac{14}{\sqrt{5}}\right) \end{pmatrix}$$

per cui

$$c_1 = \begin{pmatrix} -1 \\ -\sqrt{5} \end{pmatrix}, \quad c_2 = \begin{pmatrix} \frac{2}{3}\left(2 + \frac{7}{\sqrt{5}}\right) \\ \frac{2}{3}\left(-1 + \frac{14}{\sqrt{5}}\right) \end{pmatrix}.$$

Sostituendo all'indietro si ottiene

$$Rx = c_1 \Rightarrow \begin{pmatrix} -2 & -3 \\ 0 & -\sqrt{5} \end{pmatrix} x = \begin{pmatrix} -1 \\ -\sqrt{5} \end{pmatrix} \Rightarrow x = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

e il residuo risulta

$$\|r\|_2 = \|c_2\|_2 = 2\sqrt{6}$$

che conferma quanto ottenuto con le equazioni normali.

Si osservi come entrambi i metodi proposti risolvano alla fine lo stesso sistema triangolare, infatti $R = -L^T$. Inoltre, come detto, non si è calcolata esplicitamente la matrice Q. Se questa dovesse servire per altre computazioni è sufficiente ricostruirla dai vettori v_1 e v_2.

In MATLAB le istruzioni per il metodo delle equazioni normali sono

```
>>A=[1,0;1,1;1,2;1,3],b=[1;-4;3;2]
A=
     1     0
     1     1
     1     2
     1     3
b=
     1
    -4
     3
     2
>>A'*A,A'*b
ans=
     4     6
     6    14
ans=
     2
     8
>>L=chol(A'*A);L=L'
L=
   2.0000         0
   3.0000    2.2361
>>y=L\(A'*b)
y=
   1.0000
   2.2361
>>x=L'\y
x=
```

```
   -1
    1
>>r=norm(b-A*x,2)
r=
   4.8990
```

mentre per la fattorizzazione QR

```
>>[Q,R]=qr(A)
Q=
  -0.5000    0.6708    0.0236    0.5472
  -0.5000    0.2236   -0.4393   -0.7120
  -0.5000   -0.2236    0.8079   -0.2176
  -0.5000   -0.6708   -0.3921    0.3824
R=
  -2.0000   -3.0000
        0   -2.2361
        0         0
        0         0
>>c=Q'*b
c=
  -1.0000
  -2.2361
   3.4203
   3.5073
>>x=R(1:2,1:2)\c(1:2)
x=
   -1
    1
>>r=norm(c(3:4),2)
r=
   4.8990
```

La fattorizzazione QR è il metodo sul quale si basa l'istruzione \ quando il sistema lineare non è quadrato (altrimenti l'algoritmo è la fattorizzazione LU con pivot parziale). Basta dunque un'unico comando per ottenere la soluzione x:

```
>>x=A\b
x=
   -1
    1
```

■

Esercizio 5 *(vedi anche* 03/06/01−♯7*,* 09/12/02−♯5*,* 10/07/03−♯5*,* 24/07/03−♯6*) Dati* $m+1$ *punti* (x_i, y_i)*,* $i = 0, \ldots, m$*, e una base di* $n+1$ *funzioni modello*

ϕ_j, $j = 0, \ldots, n$, si determinino gli $n+1$ parametri a_j, $j = 0, \ldots, n$, che caratterizzano la funzione $\phi = \sum_{j=0}^{n} a_j \phi_j$ che meglio approssima i punti assegnati nel senso dei minimi quadrati, ovvero

$$\min_{a \in \mathbb{R}^{n+1}} \left(\sum_{i=0}^{m} (y_i - \phi(x_i))^2 \right)^{1/2}.$$

Soluzione. Il problema è noto come *data fitting* lineare, in quanto la funzione cercata ϕ è lineare secondo il vettore dei parametri $a = (a_0, \ldots, a_n)^T$, ovvero ϕ è combinazione lineare delle funzioni base ϕ_j, $j = 0, \ldots, n$. Esso si riconduce alla risoluzione, nel senso dei minimi quadrati, di un sistema lineare sovradeterminato quando $n < m$ (per $n = m$ si ha un sistema lineare quadrato e si parla di interpolazione, per $n > m$ si ha un sistema lineare sottodeterminato). Limitandosi, infatti, al caso sovradeterminato, si definisce la matrice $\Phi \in \mathbb{R}^{m \times (n+1)}$ di coefficienti

$$\phi_{ij} = \phi_j(x_i), \quad i = 0, \ldots, m, \quad j = 0, \ldots, n$$

e si determina il vettore dei parametri a risolvendo nel senso dei minimi quadrati il sistema lineare

$$\Phi a \approx y$$

con metodi opportuni (ad esempio la fattorizzazione QR).

La scelta delle funzioni base è fondamentale per ottenere una migliore approssimazione dei dati (x_i, y_i) assegnati e un'accurata osservazione preliminare dell'andamento di questi ultimi aiuta in tal senso: se ad esempio si ha un comportamento asintotico, la scelta di funzioni polinomiali non fornirà sicuramente un *data fitting* corretto, mentre di fronte ad un comportamento periodico si sceglierà una base di funzioni trigonometriche.

Si analizzino come casi particolari il *data fitting* polinomiale e quello esponenziale. Nel primo caso si scelgono come funzioni base i polinomi $\phi_j = x^j$, $j = 0, \ldots, n$, per cui la funzione cercata è il polinomio di grado n

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

La matrice del sistema risulta in questo caso

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{pmatrix}$$

e prende il nome di matrice di *Vandermonde*. Si dimostra che se i punti sono distinti, V ha rango pieno uguale ad $n+1$, per cui il problema ammette un'unica soluzione a ed il polinomio p_n è univocamente determinato.

Nel caso esponenziale, invece, si scelgono come funzioni base gli esponenziali $\phi_j = e^{jx}$, $j = 0, \ldots, n$, per cui la funzione cercata è la combinazione esponenziale

$$e_n(x) = a_0 + a_1 e^x + a_2 e^{2x} + \cdots + a_n e^{nx}.$$

La matrice del sistema risulta in questo caso

$$E = \begin{pmatrix} 1 & e^{x_0} & e^{2x_0} & \cdots & e^{nx_0} \\ 1 & e^{x_1} & e^{2x_1} & \cdots & e^{nx_1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & e^{x_m} & e^{2x_m} & \cdots & e^{nx_m} \end{pmatrix}$$

ed ha anch'essa rango pieno uguale ad $n+1$ se i punti sono distinti. In particolare se si scelgono come basi gli esponenziali complessi $\phi_j = e^{jx\mathrm{i}}$, $j = 0, \ldots, n$, si ottiene una funzione trigonometrica in quanto $e^{jx\mathrm{i}} = \sin jx + \mathrm{i}\cos jx$, di cui si considera la parte reale o quella immaginaria.

Come esempio applicativo in MATLAB si considerino i seguenti dati sulla popolazione statunitense dal 1790 al 1990 in milioni di persone:

```
>>load census
>>[cdate,pop]
ans=
   1.0e+003 *

   1.7900    0.0039
   1.8000    0.0053
   1.8100    0.0072
   1.8200    0.0096
   1.8300    0.0129
   1.8400    0.0171
   1.8500    0.0231
   1.8600    0.0314
   1.8700    0.0386
   1.8800    0.0502
   1.8900    0.0629
   1.9000    0.0760
   1.9100    0.0920
   1.9200    0.1057
   1.9300    0.1228
   1.9400    0.1317
   1.9500    0.1507
   1.9600    0.1790
   1.9700    0.2050
   1.9800    0.2265
   1.9900    0.2487
```

Per il *data fitting* polinomiale il MATLAB fornisce le function *polyfit* e *polyval* che determinano i coefficienti del polinomio approssimante di grado n (risolvendo il sistema di *Vandermonde*) e lo valutano in determinati punti (si veda l'*help* di MATLAB). È utile in questo particolare caso centrare rispetto alla media e riscalare rispetto alla deviazione standard i valori degli anni: essendo infatti dell'ordine di 10^3, darebbero problemi di *scaling* nella costruzione della

matrice di *Vandermonde*, risultando in un polinomio dei minimi quadrati mal condizionato. Per $n = 4$ si ha ad esempio

```
>>p=polyfit(cdate,pop,4)
Warning: Polynomial is badly conditioned. Remove repeated data
points or try centering and scaling as described in HELP POLYFIT.
p =
  1.0e+005 *
    0.0000   -0.0000    0.0000   -0.0126    6.0020
```

riscalando invece si ottiene

```
>> sdate=(cdate-mean(cdate))./std(cdate);
>> p=polyfit(sdate,pop,4)
p =
    0.7047    0.9210   23.4706   73.8598   62.2285
>>pop4=polyval(p,sdate);
>>r=pop4-pop;
>>[cdate,pop,pop4,r]
ans=
  1.0e+003 *
    1.7900    0.0039    0.0051    0.0012
    1.8000    0.0053    0.0048   -0.0005
    1.8100    0.0072    0.0060   -0.0012
    1.8200    0.0096    0.0086   -0.0010
    1.8300    0.0129    0.0125   -0.0004
    1.8400    0.0171    0.0178    0.0007
    1.8500    0.0231    0.0242    0.0011
    1.8600    0.0314    0.0319    0.0005
    1.8700    0.0386    0.0408    0.0022
    1.8800    0.0502    0.0509    0.0007
    1.8900    0.0629    0.0622   -0.0007
    1.9000    0.0760    0.0747   -0.0013
    1.9100    0.0920    0.0885   -0.0035
    1.9200    0.1057    0.1036   -0.0021
    1.9300    0.1228    0.1200   -0.0028
    1.9400    0.1317    0.1378    0.0061
    1.9500    0.1507    0.1570    0.0063
    1.9600    0.1790    0.1779   -0.0011
    1.9700    0.2050    0.2004   -0.0046
    1.9800    0.2265    0.2247   -0.0018
    1.9900    0.2487    0.2508    0.0021
```

Il *data fitting* esponenziale non è invece gestito da MATLAB, per cui l'utente deve costruire direttamente la matrice E e poi risolvere il sistema sovradeterminato tramite l'operatore \. Si ottiene, sempre per $n = 4$:

```
>>[cdate,pop,pope4,re]
```

```
ans=
  1.0e+003 *
    1.7900    0.0039    0.0081    0.0042
    1.8000    0.0053   -0.0067   -0.0120
    1.8100    0.0072    0.0081    0.0009
    1.8200    0.0096    0.0188    0.0092
    1.8300    0.0129    0.0214    0.0085
    1.8400    0.0171    0.0201    0.0030
    1.8500    0.0231    0.0194   -0.0037
    1.8600    0.0314    0.0223   -0.0091
    1.8700    0.0386    0.0297   -0.0089
    1.8800    0.0502    0.0415   -0.0087
    1.8900    0.0629    0.0569   -0.0060
    1.9000    0.0760    0.0747   -0.0013
    1.9100    0.0920    0.0939    0.0019
    1.9200    0.1057    0.1136    0.0079
    1.9300    0.1228    0.1331    0.0103
    1.9400    0.1317    0.1518    0.0201
    1.9500    0.1507    0.1693    0.0186
    1.9600    0.1790    0.1855    0.0065
    1.9700    0.2050    0.2002   -0.0048
    1.9800    0.2265    0.2134   -0.0131
    1.9900    0.2487    0.2252   -0.0235
```

Gli andamenti dell'approssimazione e del residuo sono rappresentati in Figura 19. Si osserva come l'approssimazione polinomiale produca un residuo inferiore (in valore assoluto) a quello dell'esponenziale. Ciò è dovuto al fatto che la scelta della base di polinomi risulta migliore degli esponenziali rispetto ai dati assegnati.

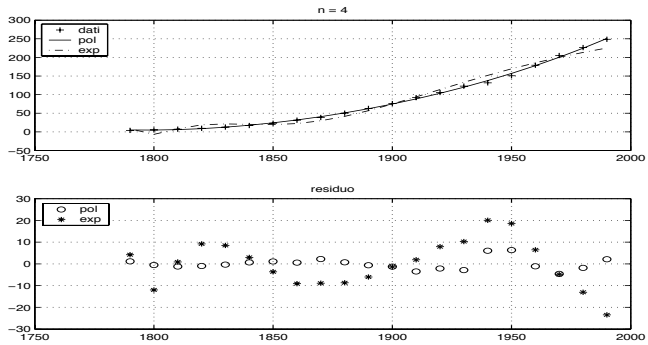


Figura 19: *Data fitting* polinomiale ed esponenziale per $n = 4$.

Si può poi valutare come varia la norma-2 del residuo all'aumentare del grado n: i risultati sono rappresentati in Figura 20. Si osserva come $\|r\|_2$ diminuisca al crescere di n per il caso polinomiale. In particolare per $n = 20$ si ha in

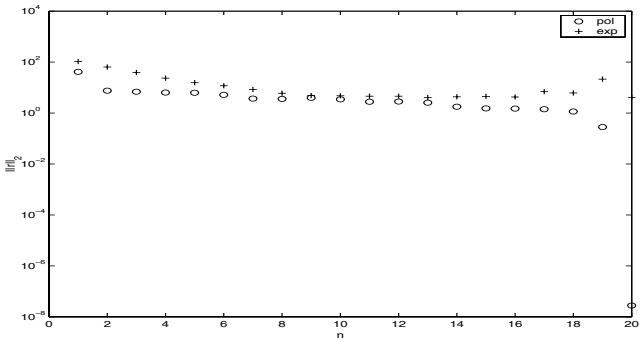


Figura 20: Norma-2 del residuo per $n = 1, \ldots, 20$.

teoria un residuo nullo in quanto esiste ed è unico il polinomio di grado $n = 20$ che passa esattamente per gli $m + 1 = 21$ dati assegnati (i.e. interpolazione). In tal caso il sistema è quadrato e la matrice di *Vandermonde* è invertibile. Nel caso esponenziale, invece, il residuo tende ad aumentare per $n \geq 14$ in quanto la matrice E del sistema è prossima ad avere rango non pieno (*nearly rank deficient*) per effetto della rappresentazione di macchina. In questo caso il MATLAB provvede ad avvisare l'utente con un messaggio a video. ■

Riferimenti bibliografici

[QS02] **testo consigliato:** Quarteroni, A., Saleri, F. (2002) *Introduzione al Calcolo Scientifico*, Springer, Milano.

[BBCM92] Bevilacqua, R., Bini, D., Capovani, M. e Menchi, O. (1992) *Metodi Numerici*, Zanichelli, pagg. 352-365, 370-374, 436-446 (senza dimostrazioni).

[Com95] Comincioli, V. (1998) *Analisi numerica. Metodi, modelli, applicazioni*, McGraw-Hill, Milano, pagg. 112115, 122, 125-126,132-136, 139-140,150-155

[Mon98] Monegato, G. (1998) *Fondamenti di Calcolo Numerico*, Clut, pagg. 121-143, 154-162, 170-172.

[NPR01] Naldi, G., Pareschi, L. e Russo, G. (2001) *Introduzione al Calcolo Scientifico*, Mc Graw Hill Ed., pagg. 133-154,167-189.

[Hea02] Heath, M.T. (2002) *Scientific Computing*, Mc Graw Hill Ed., pagg. 309-332 (senza dimostrazioni).

[Hi93] Higham, N. J. (1993) *The accuracy of floating point summation*, SIAM J. Sci. Comput. 14, 783-799.

[Ste96] Stewart, G.W. (1996) *Afternotes on Numerical Analysis*, SIAM Ed., pagg. 31-38, 79-97.