Matematica utile

definizioni, formule ed esempi

Pietro Barbiero

Quest'opera contiene informazioni tratte da wikipedia (http://www.wikipedia.en) e dalle dispense relative al corso di Analisi Matematica II tenuto dalla professoressa Scuderi Letizia del Dipartimento di Scienze Matematiche del Politecnico di Torino (IT).

Indice

Parte I

Formulario di Calcolo Numerico

Capitolo 1

Aritmetica

1.1 Numeri macchina

DEF 1.1 (rappresentazione in virgola mobile (floating point) di un numero reale)**.** Metodo di rappresentazione approssimata dei numeri reali utile alla loro elaborazione nei sistemi discreti (computer):

$$a = pN^q \tag{1.1}$$

p: numero reale
N: base del sistema di numerazione
q: numero intero
e.g.: se $N = 10$ e $a = 0.015 \implies a = 0.00015 \cdot 10^2$

DEF 1.2 (rappresentazione floating point normalizzata)**.** Rappresentazione univoca definita dalla relazione (in un computer è sufficiente memorizzare p e q):

$$N^{-1} \leq |p| < 1 \tag{1.2}$$

per $N = 10$

$$0.1 \leq |p| < 1$$

p: mantissa di a
q: esponente o caratteristica di a
e.g.: se $N = 10$ e $a = 0.015 \implies a = 0.15 \cdot 10^{-1}$ con $p = 0.15$ e $q = -1$

Limiti della rappresentazione in virgola mobile

OSS 1 (limite sulla mantissa). La mantissa p può avere al massimo t cifre

OSS 2 (limite sull'esponente). Dati due interi $m < 0$ e $M > 0$ l'esponente $m \leq q \leq M$

DEF 1.3 (numero macchina)**.** Numero in virgola mobile la cui mantissa e il cui esponente sono esattamente rappresentabili negli spazi a loro riservati:

$$n_{cifre-di-p} \leq t \wedge m \leq q \leq M \tag{1.3}$$

e.g.: in una macchina in cui $N = 10$, $t = 5$, $m = -127$ e $M = 128$:
$a = 1.58292 \cdot 10^{128} = 0.158292 \cdot 10^{129}$ non è un numero macchina ($p > t$ e $q > M$)
$a = 0.0038245 \cdot 10^{130} = 0.38245 \cdot 10^{128}$ è un numero macchina

1.2 Errori

DEF 1.4 (errore di underflow)**.** Errore che si commette quando l'esponente di un numero in virgola mobile è: $q < m$ (rende impossibile la rappresentazione)

DEF 1.5 (errore di overflow)**.** Errore che si commette quando l'esponente di un numero in virgola mobile è: $q > M$ (rende impossibile la rappresentazione)

DEF 1.6 (errore di troncamento)**.** Errore che si commette quando si vuole rappresentare un numero in virgola mobile con $n_{cifre-di-p} > t$: si escludono le cifre che seguono la t-esima cifra
e.g.: per $N = 10$ e $t = 5$: $a = 0.158295 \cdot 10^1 \implies \bar{a} = 0.15829 \cdot 10^1$

DEF 1.7 (errore di arrotondamento)**.** Errore che si commette quando si vuole rappresentare un numero in virgola mobile con $n_{cifre-di-p} > t$: si aggiunge $\frac{1}{2}N^{-t}$ a p e si tronca il risultato alla t-esima cifra
e.g.: per $N = 10$ e $t = 5$: $a = 0.158295 \cdot 10^1 \implies \bar{a} = 0.158295 \cdot 10^1 + 0.5 \cdot 10^{-5} = 0.158300 \cdot 10^1 = 0.15830 \cdot 10^1$

DEF 1.8 (errore assoluto)**.** Errore definito come la differenza tra il numero macchina $\bar{a}$ e il numero reale a:

$$e_a = |a - \bar{a}| \tag{1.4}$$

OSS 3. Per $a = p \cdot N^q$, $p \in (\bar{p_1}, \bar{p_2})$, $\bar{p_1} \cdot \bar{p_2} > 0$ e $\bar{p_2} - \bar{p_1} = N^{-t}$:

- troncando: $\bar{p} = \bar{p_1} \implies |p - \bar{p}| < N^{-t} \implies e_a < N^{q-t}$
- arrotondando: $\begin{cases} p \in (\bar{p_1}, \bar{p_1} + \frac{1}{2}N^{-t}) \implies \bar{p} = \bar{p_1} \\ p \in (\bar{p_1} + \frac{1}{2}N^{-t}, \bar{p_2}) \implies \bar{p} = \bar{p_2} \end{cases} \implies |p - \bar{p}| \leq \frac{1}{2}N^{-t} \implies e_a \leq \frac{1}{2}N^{q-t}$

e.g.: per $a = 0.158295 \cdot 10^1$ e $0.15830 - 0.15829 = 10^{-5} = 0.00001$:

- troncando: $e_a = |0.158295 \cdot 10^1 - 0.15829 \cdot 10^1| = 0.000005 \cdot 10^1 = 0.5 \cdot 10^{-5} < 10^{1-5} = 10^{-4}$
- arrotondando: $e_a = |0.158295 \cdot 10^1 - 0.15830 \cdot 10^1| = 0.000005 \cdot 10^1 = 0.5 \cdot 10^{-5} < 0.5 \cdot 10^{1-5} = = 0.5 \cdot 10^{-4}$

DEF 1.9 (errore relativo)**.** Errore definito come il rapporto tra l'errore assoluto e_a e il valore assoluto del numero reale a:

$$e_r = \frac{e_a}{|a|} = \frac{|a - \bar{a}|}{|a|} \implies \bar{a} = a(1 + e_r) \tag{1.5}$$

OSS 4. Siccome $|p| \geq N^{-1} \implies |a| \geq N^{q-1} \implies e_r \leq \frac{e_a}{N^{q-1}}$:

- troncando: $e_r < \frac{N^{q-t}}{N^{q-1}} = N^{q-t} \cdot N^{1-q} = N^{1-t}$
- arrotondando: $e_r \leq \frac{\frac{1}{2}N^{q-t}}{N^{q-1}} = \frac{1}{2} \cdot N^{q-t} \cdot N^{1-q} = \frac{1}{2}N^{1-t}$

DEF 1.10 (precisione di macchina o unità di roundoff)**.** costante caratteristica di ogni aritmetica in virgola mobile che rappresenta una misura della massima precisione di calcolo raggiungibile:

$$eps = \min\{\bar{a} \in F \wedge \bar{a} > 0 \mid 1 \oplus \bar{a} > 1\} \tag{1.6}$$

OSS 5.

$$eps = \begin{cases} N^{1-t} \text{ per la tecnica di troncamento} \\ \frac{1}{2}N^{1-t} \text{ per la tecnica di arrotondamento} \end{cases} \tag{1.7}$$

OSS 6. Per l'osservazione OSS 4: $e_r \leq eps$

DEF 1.11 (operazione di macchina)**.** Operazione che associa a due numeri di macchina un terzo numero di macchina:

$$\bar{a_1} \odot \bar{a_2} = (\bar{a_1} \cdot \bar{a_2})(1 + e_{r\odot}) \tag{1.8}$$

OSS 7. Vale la proprietà commutativa della somma di macchina ($\oplus$) e del prodotto di macchina ($\otimes$) (in generale non valgono né la proprietà associativa né la proprità distributiva)

DEF 1.12 (cancellazione numerica)**.** Perdita di cifre della mantissa dovuta ad una sottrazione tra due numeri "quasi uguali" arrotondati:

$$\bar{a_1} \ominus \bar{a_2} \text{ e } \bar{a_1} \approx \bar{a_2} \tag{1.9}$$

e.g.: per $N = 10$, $t = 5$, $a_1 = 0.15723944$ e $a_2 = 0.15722131$:
in aritmetica esatta: $a_1 - a_2 = 0.15723944 - 0.15722131 = 0.00001813 = 0.1813 \cdot 10^{-4}$
in aritmetica finita: $\bar{a_1} \ominus \bar{a_2} = 0.15723 - 0.15722 = 0.00001 = 0.10000 \cdot 10^{-4}$

1.3 Problemi numerici

DEF 1.13 (problema numerico)**.** Descrizione non ambigua di una connessione funzionale f:

$$x \underset{f}{\rightarrow} y \tag{1.10}$$

tra i dati x (input) e i risultati y (output)

OSS 8. La connessione $\underset{f}{\rightarrow}$ può essere:

- esplicita: $y = f(x)$ (e.g.: $y = 3x$)
- implicita: $f(x, y) = 0$ (e.g.: $f(0) = 0$ e $f(1) = 3$)

DEF 1.14 (condizionamento di un problema numerico)**.** Rapporto tra l'errore commesso sul risultato di un calcolo e l'incertezza sui dati di ingresso. Se si indicano con x i dati in ingresso, con $\bar{x}$ una loro perturbazione e con $f(x)$ e $f(\bar{x})$ i corrispondenti risultati in precisione finita di calcolo:

$$\frac{||f(x) - f(\bar{x})||}{||f(x)||} \begin{cases} \approx K(f, x)\frac{||x-\bar{x}||}{||x||} \\ \leq K(f, x)\frac{||x-\bar{x}||}{||x||} \end{cases} \tag{1.11}$$

dove $K(f, x)$ prende il nome di numero di condizionamento

OSS 9. Se $K(f, x) \approx 1$ il problema è ben condizionato, altrimenti il problema è mal condizionato

e.g.: per $y = x_1 + x_2$:
$\frac{||f(x)-f(\bar{x})||}{||f(x)||} = \frac{x_1+x_2-(\bar{x_1}+\bar{x_2})}{x_1+x_2} = \frac{x_1+x_2-x_1(1+e_{r1})-x_2(1+e_{r2})}{x_1+x_2} = -\frac{x_1}{x_1+x_2}e_{r1} - \frac{x_2}{x_1+x_2}e_{r2} \implies K_i = \left|\frac{x_i}{x_1+x_2}\right|$
se $(x_1 + x_2) \to 0 \implies K_i \to \infty$ e quindi il problema è mal condizionato

DEF 1.15 (algoritmo numerico)**.** Sequenza finita di operazioni che consente di ottenere l'output $y*$ del problema $y = f(x)$

DEF 1.16 (stabilità numerica di un algoritmo)**.** Un algoritmo è numericamente stabile se l'errore relativo associato al risultato $y*$ ha lo stesso ordine di grandezza della precisione di macchina *eps*:

$$\frac{||f(\bar{x}) - y*||}{||f(\bar{x})||} \approx eps \tag{1.12}$$

dove $y*$ è il risultato fornito dall'algoritmo (a partire dai dati $\bar{x}$ perturbati dall'arrotondamento in precisione finita di calcolo) e $f(\bar{x})$ è la soluzione del problema (a partire dai dati $\bar{x}$ in precisione infinita di calcolo)

Capitolo 2

Sistemi di equazioni lineari

2.1 Vettori

2.1.1 Norme di un vettore

DEF 2.1 (norma di un vettore)**.** Funzione che associa ad ogni vettore di uno spazio vettoriale una lunghezza positiva:

$$\begin{aligned} ||\cdot|| : \quad & X \to [0, +\infty) \\ & x \to ||x|| \end{aligned} \tag{2.1}$$

2.1.1.1 Norma 1

DEF 2.2 (norma 1)**.** Somma dei valori assoluti dei valori assoluti dei componenti di un vettore (generalizzazione vettoriale del valore assoluto):

$$||\vec{x}||_1 := \sum_{i=1}^{n} |x_i| \tag{2.2}$$

e.g.: per $\vec{x} = (1, 2, 3, 4)^T$: $||\vec{x}||_1 = 1 + 2 + 3 + 4 = 10$

MATLAB 1. Codice Matlab

```
1 v = [1,2,3,4];
2 norm(v,1)
```

ans = 10

2.1.1.2 Norma 2

DEF 2.3 (norma 2 o euclidea)**.** Radice della somma dei quadrati delle componenti di un vettore (o radice del prodotto scalare tra il vettore e il vettore trasposto) (generalizzazione del teorema di Pitagora):

$$||\vec{x}||_2 := \sqrt{\sum_{i=1}^{n} x_i^2} \quad \text{oppure} \quad ||\vec{x}||_2 := \sqrt{\vec{x}^T \cdot \vec{x}} \tag{2.3}$$

e.g.: per $\vec{x} = (1, 2, 3, 4)^T$: $||\vec{x}||_2 = \sqrt{1^2 + 2^2 + 3^2 + 4^2} = 5.4772$

MATLAB 2. Codice Matlab

```
1 v = [1,2,3,4];
2 norm(v,2)
```

ans = 5.4772

2.1.1.3 Norma ∞

DEF 2.4 (norma infinito)**.** Massimo dei componenti di un vettore in valore assoluto:

$$||\vec{x}||_\infty := \max_{1\leq i\leq n} |x_i| \tag{2.4}$$

e.g.: per $\vec{x} = (1,2,3,4)^T$: $||\vec{x}||_\infty = \max_{1\leq i\leq 4}\{|1|,|2|,|3|,|4|\} = 4$

MATLAB 3. Codice Matlab

```
1 v = [1,2,3,4];
2 norm(v,inf)
```

ans = 4

2.2 Matrici quadrate

2.2.1 Norme di una matrice quadrata

2.2.1.1 Norma 1

DEF 2.5 (norma 1)**.** Massima somma delle componenti in valore assoluto delle colonne di una matrice:

$$||A||_1 := \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}| \tag{2.5}$$

e.g.: per $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$: $||A||_1 = \max_{1\leq j\leq 2}\{|1|+|3|, |2|+|4|\} = 6$

MATLAB 4. Codice Matlab

```
1 A = [1,2;3,4];
2 norm(A,1)
```

ans = 6

DEF 2.6 (raggio spettrale di una matrice)**.** Massimo tra gli autovalori in valore assoluto di una matrice:

$$\rho(A) := \max_{1\leq i\leq n} |\lambda_i| \tag{2.6}$$

e.g.: per $A = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}$: $p_A(t) = |A-tI| = \begin{vmatrix} 1-t & 1 \\ 2 & 1-t \end{vmatrix} = (1-t)(1-t)-2\cdot 1 = t^2-2t-1 = 0$; quindi gli autovalori sono: $\lambda_i = \{1-\sqrt{2}, 1+\sqrt{2}\}$ e il raggio spettrale è: $\rho(A) = max_{1\leq i\leq 2}\{|1-\sqrt{2}|, |1+\sqrt{2}|\} = = 1+\sqrt{2} = 2.4142$

MATLAB 5. Codice Matlab

```
1 A = [1,1;2,1];
2 rho = max(abs(eig(A)))
```

ans = 2.4142

2.2.1.2 Norma 2

DEF 2.7 (norma 2 o norma spettrale)**.** Radice del raggio spettrale del prodotto matriciale della matrice per la sua trasposta:

$$||A||_2 := \sqrt{\rho(A^T A)} \tag{2.7}$$

e.g.: per $A = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$: $A^T A = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 4 \end{pmatrix}$; la cui caratteristica è: $p_{A^T A}(t) = \begin{vmatrix} 2-t & 2 \\ 2 & 4-t \end{vmatrix} = t^2 - 6t + 4 = 0$; i cui autovalori sono: $\lambda_i = \{3 - \sqrt{5}, 3 + \sqrt{5}\}$; il cui raggio spettrale è: $\rho(A^T A) = 3 + \sqrt{5}$; la cui norma spettrale è: $||A||_2 = \sqrt{3 + \sqrt{5}} = 2,2882\ldots$

MATLAB 6. Codice Matlab

```
1 A = [1,1;2,0];
2 norm(A,2)
```

ans = 2.2882

2.2.1.3 Norma ∞

DEF 2.8 (norma infinito)**.** Massima somma delle componenti in valore assoluto delle righe di una matrice:

$$||A||_\infty := \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| \tag{2.8}$$

e.g.: per $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$: $||A||_\infty = \max_{1 \le i \le 2}\{|1| + |2|, |3| + |4|\} = 7$

MATLAB 7. Codice Matlab

```
1 A = [1,2;3,4];
2 norm(A,inf)
```

ans = 7

2.2.1.4 Proprietà delle norme matriciali

Proprietà delle norme matriciali
$||AB|| \le ||A||||B||$
$||I|| = 1$
$||A\vec{x}|| \le ||A||||\vec{x}||$

2.2.2 Matrici quadrate particolari

2.2.2.1 Matrice triangolare

DEF 2.9 (matrice quadrata triangolare)**.** Matrice quadrata che ha tutti gli elementi nulli sotto (superiore) o sopra (inferiore) la diagonale principale:

$$A: \quad i \lesseqgtr j \implies a_{ij} = 0 \tag{2.9}$$

e.g.: la matrice $A = \begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}$ è triangolare inferiore

2.2.2.2 Matrice diagonale

DEF 2.10 (matrice quadrata diagonale)**.** Matrice quadrata in cui solo i valori della diagonale principale possono essere diversi da 0:

$$A: \quad i \neq j \implies a_{ij} = 0 \tag{2.10}$$

e.g.: la matrice $A = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$ è diagonale

2.2.2.3 Matrice a diagonale dominante

DEF 2.11 (matrice a diagonale dominante)**.** Matrice quadrata i cui elementi diagonali sono maggiori in valore assoluto alla somma di tutti i restanti elementi della stessa riga (dominante per righe) o della stessa colonna (dominante per colonne):

$$A: \quad |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \text{dominante per righe} \tag{2.11}$$

$$A: \quad |a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}| \quad \text{dominante per colonne} \tag{2.12}$$

e.g.: la matrice $A = \begin{pmatrix} 4 & 2 & 1 \\ 0 & 4 & 2 \\ 5 & 0 & 6 \end{pmatrix}$ è a diagonale dominante per righe ma non per colonne

2.2.2.4 Matrice simmetrica definita positiva

DEF 2.12 (matrice simmetrica definita positiva)**.** Matrice simmetrica i cui autovalori sono tutti reali e positivi:

$$A: \quad \forall \lambda_i \in \mathbb{R}^+ \tag{2.13}$$

OSS 10. Condizioni per cui una matrice è simmetrica definita positiva:

- $A^T = A$
- $x^T A x > 0 \quad \forall x \neq o$

e.g.: per la matrice simmetrica $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$ gli autovalori sono: $p_A(\lambda) = \begin{vmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{vmatrix} = \lambda^2 - 6\lambda + 8 = 0 \rightarrow \lambda_i = \{2, 4\}$ e quindi la matrice è definita positiva

2.3 Sistemi lineari

2.3.1 Definizioni

2.3.1.1 Sistema di equazioni lineari

DEF 2.13 (sistema di equazioni lineari)**.** Sistema di equazioni lineari che devono essere verificate tutte contemporaneamente:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \iff \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{2.14}$$

2.3.1.2 Condizionamento di un sistema lineare

PROP 2.3.1 (condizionamento nella risoluzione numerica di un sistema lineare)**.** *Se* $e_a(A) < \frac{1}{||A^{-1}||}$ *allora:*

$$e_r(x) \leq \frac{K(A)}{1 - K(A)e_r(A)}\big(e_r(A) + e_r(b)\big) \tag{2.15}$$

dove il numero di condizionamento del sistema $Ax = b$ *è* $K(A) = ||A||||A^{-1}|| \geq ||AA^{-1}|| = ||I|| = 1$ *(per le norme* 1*,* 2 *e* ∞*). Se si suppone* $e_a(A) < \frac{1}{2||A^{-1}||}$ *allora:*

$$e_r(x) \leq 2K(A)\big(e_r(A) + e_r(b)\big) \tag{2.16}$$

OSS 11. Se $K(A) \approx 1$ il sistema è ben condizionato, altrimenti se $K(A) \gg 1$ il sistema è mal condizionato

MATLAB 8. Codice Matlab

```
1 A = [1,2;3,4];
2 cond(A,1)
```

ans = 21

OSS 12. Esempi di sistemi mal condizionati

- Matrice di Hilbert
- Matrice di Vandermonde

2.3.2 Algoritmi per la risoluzione di sistemi lineari

2.3.2.1 Algoritmi diretti

2.3.2.1.1 Algoritmo di sostituzione all'indietro

ALG 2.1 (algoritmo di sostituzione all'indietro). Metodo di risoluzione di sistemi lineari triangolari superiori:

Algorithm 2.1 Algoritmo di sostituzione all'indietro (A triangolare superiore) $[O(n^2)]$

1: $x(n) = \frac{b(n)}{A(n,n)}$ // Calcola l'n-esima soluzione
2: **for** $(i = n-1; i \geq 1; i-1)$ // Per ogni riga a partire dalla penultima fino alla prima
3: $s = 0$ // Inizializza la variabile somma a 0
4: **for** $(j = i+1; j \leq n; j+1)$ // Per ogni colonna a partire da quella che non contiene l'incognita fino all'ultima
5: $s = s + A(i,j)x(j)$ // Somma tutti gli elementi di una riga
6: **for end**
7: $x(i) = \frac{(b(i)-s)}{A(i,i)}$ // Calcola l'i-esima soluzione
8: **for end**

OSS 13. Costo computazionale: $O(n^2)$

MATLAB 9. Codice Matlab

```
A = [1 2 3; 0 3 4; 0 0 5];
b = [12; 23; 10];
n = length(b);
x = zeros(n,1);

x(n) = b(n)/A(n,n);
for i = n-1:-1:1
    s = 0;
    for j = i+1:1:n
        s = s+A(i,j)*x(j);
    end
    x(i) = (b(i)-s)/A(i,i);
end
```

$x = (-4, 5, 2)$

2.3.2.1.2 Algoritmo di sostituzione in avanti

ALG 2.2 (algoritmo di sostituzione in avanti). Metodo di risoluzione di sistemi lineari triangolari inferiori:

Algorithm 2.2 Algoritmo di sostituzione in avanti (A triangolare inferiore) $[O(n^2)]$

1: $x(1) = \frac{b(1)}{A(1,1)}$ // Calcola la prima soluzione
2: **for** $(i = 2; i \leq n; i + 1)$ // Per ogni riga a partire dalla seconda fino all'ultima
3: $s = 0$ // Inizializza la variabile somma a 0
4: **for** $(j = 1; j \leq i - 1; j + 1)$ // Per ogni colonna a partire dalla prima fino a quella che contiene l'incognita
5: $s = s + A(i, j)x(j)$ // Somma tutti gli elementi di una riga
6: **for end**
7: $x(i) = \frac{(b(i)-s)}{A(i,i)}$ // Calcola l'i-esima soluzione
8: **for end**

OSS 14. Costo computazionale: $O(n^2)$

MATLAB 10. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];
b = [2; 7; 5];
n = length(b);
x = zeros(n,1);

x(1) = b(1)/A(1,1);
for i = 2:n
    s = A(i,1:i-1)*x(1:i-1);
    x(i) = (b(i)-s)/A(i,i);
end
```

$x = (2, 1, -1)$

2.3.2.1.3 Algoritmo di Gauss

ALG 2.3 (algoritmo di Gauss). Metodo di risoluzione di qualunque sistema lineare $Ax = b$ non singolare:

Algorithm 2.3 Algoritmo di Gauss $[O(n^3/3)]$

for $(k = 1; k \leq n-1; k+1)$ // Per ogni colonna dalla prima fino alla penultima
  for $(i = k+1; i \leq n; i+1)$ // Per ogni riga successiva alla riga corrispondente alla colonna individuata da k
    $A(i,k) = -\frac{A(i,k)}{A(k,k)}$ // Memorizzo nella matrice il moltiplicatore che azzera il k-esimo elemento della i-esima riga
    for $(j = k+1; j \leq n; j+1)$ // Per ogni colonna della i-esima riga
      $A(i,j) = A(i,j) + A(i,k)A(k,j)$ // Sommo a ciascun elemento della i-esima riga il prodotto
      // tra il moltiplicatore e l'elemento corrispondente della riga precedente
    for end
    $b(i) = b(i) + A(i,k)b(k)$ // Sommo all'i-esimo elemento del vettore b il prodotto
    // tra il moltiplicatore e il k-esimo elemento di b
  for end
for end

OSS 15. Costo computazionale: $O(n^3/3)$

OSS 16. Per procedere con il metodo di Gauss è necessario che $\forall k \quad A(k,k) \neq 0$. Tale condizione è garantita se vale almeno una delle seguenti proprietà:

- $det(A_k) \neq$ (A_k: matrice di ordine k)
- A è a diagonale dominante per righe
- A è a diagonale dominante per colonne
- A è simmetrica definita positiva

Se $A(k,k) = 0$ occorre scambiare la k-esima riga con l'i-esima riga per cui $A(i,k) \neq 0$ (sempre possibile se A è non singolare)

DEF 2.14 (pivoting parziale). Per garantire una maggiore stabilità numerica all'algoritmo di Gauss conviene scambiare la k-esima riga con la riga r-esima:

$$|A(r,k)| = \max_{k \leq i \leq n} |A(i,k)| \tag{2.17}$$

Il pivoting parziale è superfluo quando:

- A è a diagonale dominante per colonne
- A è simmetrica a diagonale dominante
- A è simmetrica definita positiva

MATLAB 11. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];
b = [2; 7; 5];
n = length(b);
x = zeros(n,1);

%%% algoritmo di Gauss %%%
for k = 1:n-1
    A(k+1:n,k) = -A(k+1:n,k)/A(k,k);
```

```
    A(k+1:n,k+1:n) = A(k+1:n,k+1:n)+A(k+1:n,k)*A(k,k+1:n);
    b(k+1:n) = b(k+1:n)+A(k+1:n,k)*b(k);
end

%%% algoritmo di sostituzione all'indietro %%%
x(n) = b(n)/A(n,n);
for i = n-1:-1:1
    s = A(i,i+1:n)*x(i+1:n)
    x(i) = (b(i)-s)/A(i,i);
end
```

$x = (2, 1, -1)$

oppure:

MATLAB 12. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];
b = [2; 7; 5];

x = A\b
```

$x = (2, 1, -1)$

DEF 2.15 (fattorizzazione di Gauss). Ridefinizione dell'algoritmo di Gauss utilizzando le matrici P (di permutazione) per scambiare le righe e le matrici M per la memorizzazione dei moltiplicatori:

$$\begin{aligned} M_{n-1}P_{n-1}\dots M_1P_1Ax &= M_{n-1}P_{n-1}\dots M_1P_1b \\ GAx &= Gb \\ Ux &= \bar{b} \end{aligned} \tag{2.18}$$

Riordinando opportunamente i fattori della decomposizione $GA = U$ si ottiene:

$$\begin{aligned} \bar{M_{n-1}}\dots\bar{M_1}P_{n-1}\dots P_1A &= U \\ \bar{M}PA &= U \end{aligned} \tag{2.19}$$

Indicando con $L = \bar{M}^{-1}$ si ottiene la fattorizzazione:

$$PA = LU \tag{2.20}$$

dove L è una matrice triangolare inferiore con diagonale unitaria contenente l'inverso dei moltiplicatori e U è la matrice triangolare superiore della decomposizione di Gauss

MATLAB 13. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];

[L,U,P] = lu(A)
```

$x = (2, 1, -1)$

DEF 2.16 (fattorizzazione di Choleski). Se A è simmetrica definita positiva (poiché il pivoting è superfluo) si può riscrivere la fattorizzazione di Gauss $PA = LU$ come:

$$A = L_1L_1^T \tag{2.21}$$

Dove L_1 è una matrice triangolare inferiore con elementi positivi sulla diagonale principale

OSS 17. Costo computazionale: $O(n^3/6)$

MATLAB 14. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];

R = chol(A)
```

$x = (2, 1, -1)$

APPL 1 (fattorizzazione di Gauss). Applicazioni della fattorizzazione di Gauss $PA = LU$:

- risoluzione del sistema lineare $Ax = b$:

$$Ax = b \iff PAx = Pb = LUx \tag{2.22}$$

  dove se si pone $Ux = y$:

$$\begin{cases} Ly = Pb \implies y = PbL^{-1} \\ Ux = y \implies x = yU^{-1} \end{cases} \tag{2.23}$$

- calcolo del determinante di A:

$$det(A) = (-1)^s \prod_{i=1}^{n} U(i,i) \tag{2.24}$$

  dove s è il numero totale di scambi di righe effettuati

- calcolo dell'inversa di A:

$$PA = LU \iff A^{-1}P^{-1} = U^{-1}L^{-1} \iff A^{-1} = U^{-1}L^{-1}P \tag{2.25}$$

  con un costo computazionale di $O(n^3)$

- risoluzione di p sistemi lineari aventi la matrice dei coefficienti in comune:

$$\begin{cases} Ax_1 = b_1 \\ \quad \vdots \\ Ax_p = b_p \end{cases} \tag{2.26}$$

  da cui:

$$\begin{cases} Ly_i = Pb_i \implies y_i = Pb_iL^{-1} \\ Ux_i = y_i \implies x_i = y_iU^{-1} \end{cases} \tag{2.27}$$

  con un costo computazionale di $O(n^3/3 + pn^2)$

- risoluzione del sistema non lineare $A^p x = b$:

$$A^p x = b \iff A\big(\dots(Ax)\big) = Az = A(Ay) = A(A(Aw)) = \dots \iff \begin{cases} Az = b \\ Ay = z \\ Aw = y \\ \quad \vdots \end{cases} \tag{2.28}$$

  con un costo computazionale di $O(2n^3/3)$

APPL 2 (fattorizzazione di Choleski). Applicazioni della fattorizzazione di Choleski $A = L_1 L_1^T$:

- risoluzione del sistema lineare $Ax = b$:

$$Ax = L_1 L_1^T x = L_1 y = b \iff \begin{cases} L_1 y = b \implies y = bL_1 \\ L_1^T x = y \implies x = yL_1^T \end{cases} \tag{2.29}$$

- calcolo dell'inversa di A:

$$A = L_1 L_1^T \iff A^{-1} = (L^{-1})^T L^{-1} \tag{2.30}$$

  con un costo computazionale di $O(2n^3/3)$

2.3.2.2 Definizioni

2.3.2.2.1 Metodo iterativo

DEF 2.17 (metodo iterativo)**.** Metodo di risoluzione di un sistema lineare $Ax = b$ non singolare in cui, a partire da un vettore arbitrario x_0 di dimensione n, si determina una successione di vettori $\{x_{k+1}\}_{0 \leq k \leq n}$ che, sotto opportune condizioni converge alla soluzione esatta x:

$$\lim_{k \to \infty} (x - x_k) = \vec{o} \tag{2.31}$$

Gli algoritmi di Jacobi e di Gauss-Seidel fanno parte di una classe di algoritmi iterativi per cui:

$$Ax = b \iff (D + C)x = b \iff Dx = b - Cx \iff Dx_{k+1} = b - Cx_k \tag{2.32}$$

dove alla k-esima iterazione x_{k+1} è soluzione del sistema lineare

OSS 18. La matrice D deve soddisfare le seguenti condizioni:

- non singolare
- di forma semplice (triangolare, diagonale) tale per cui la soluzione del sistema si possa ottenere con un algoritmo poco costoso
- tale da garantire la convergenza per $k \to \infty$

DEF 2.18 (tolleranza)**.** Quando si implementa un algoritmo iterativo è necessario definire dei criteri di arresto dell'iterazione:

- si può fissare un numero massimo di iterazioni
- si può fissare una tolleranza relativa $toll_r \geq eps$ in modo che l'iterazione si arresti quando:

$$||x^{\bar{k}+1} - x^{\bar{k}}|| \leq toll_r ||x^{\bar{k}+1}|| \tag{2.33}$$

- si può fissare una tolleranza assoluta $toll_a$ in modo che l'iterazione si arresti quando:

$$||x^{\bar{k}+1} - x^{\bar{k}}|| \leq toll_r \tag{2.34}$$

OSS 19. I metodi iterativi sono efficienti per matrici sparse o di grandi dimensioni

OSS 20. Nei metodi iterativi la soluzione viene determinata come limite di una successione di vettori convergente

2.3.2.2.2 Convergenza dei metodi iterativi

PROP 2.3.2 (convergenza dei metodi iterativi)**.** *Posto:*

$$e_k = x - x_k \tag{2.35}$$

e:

$$B = -D^{-1}C = -D^{-1}(A - D) = I - D^{-1}A \tag{2.36}$$

dove B viene chiamata matrice di iterazione, allora facendo la differenza tra:

$$\begin{array}{ll} Dx = b - Cx & - \\ Dx_k = b - Cx_{k-1} & = \\ \hline De_k = -Ce_{k-1} & \end{array} \tag{2.37}$$

si ottiene:

$$e_k = -D^{-1}Ce_{k-1} \implies e_k = Be_{k-1} \tag{2.38}$$

da cui si deduce che:

$$\lim_{k\to\infty} e_k = \vec{o} \iff \lim_{k\to\infty} B_k = O \tag{2.39}$$

e siccome:

$$\lim_{k\to\infty} B_k = O \iff \rho(B) = \rho(I - D^{-1}A) < 1 \tag{2.40}$$

$$\rho(A) \leq ||A|| \tag{2.41}$$

allora:

$$\lim_{k\to\infty} e_k = \vec{o} \iff \rho(I - D^{-1}A) < 1 \tag{2.42}$$

$$||I - D^{-1}A|| < 1 \implies \lim_{k\to\infty} e_k = \vec{o} \tag{2.43}$$

perciò se la norma della matrice di iterazione B è minore di 1 allora è garantita la convergenza dell'algoritmo iterativo

OSS 21. Si può dimostrare che se:

- A è a diagonale dominante per righe: $\implies ||I - D^{-1}A||_\infty < 1$
- A è a diagonale dominante per colonne: $\implies ||I - D^{-1}A||_1 < 1$

e quindi gli algoritmi di Jacobi e Gauss-Seidel convergono

OSS 22. Se A è simmetrica definita positiva allora l'algoritmo di Gauss-Seidel converge

2.3.2.3 Algoritmi iterativi

2.3.2.3.1 Algoritmo di Jacobi

ALG 2.4 (algoritmo di Jacobi). Metodo di risoluzione iterativa per un sistema lineare $Ax = b$ non singolare:

Algorithm 2.4 Algoritmo di Jacobi $[O \propto \rho(I - D^{-1}A)]$

1: $D = diag(diag(A))$ // D è una matrice diagonale avente i valori della diagonali uguali ad A
2: $C = A - D$ // C è una matrice data dalla differenza tra A e D
3: **for** $(k = 1; k \leq kmax; k + 1)$ // Per $kmax$-volte
4: $x1 = D\backslash(b - Cx0)$ // Calcolo la k-esima approssimazione della soluzione del sistema
5: **if** $(norm(x1 - x0) \leq toll)$ // Se la differenza tra la soluzione precedente e quella corrente è minore della tolleranza
6: break // L'iterazione viene interrotta
7: **if end**
8: $x0 = x1$ // Memorizzo la soluzione corrente
9: **for end**

OSS 23. Costo computazionale: dipende dal raggio spettrale: minore è il raggio spettrale più rapida è la convergenza

MATLAB 15. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];
b = [2; 7; 5];
kmax = 10;
toll = 0,01;
n = length(b);
x0 = zeros(n,1);
D = diag(diag(A));
C = A-D;

for k = 1:kmax
    x1 = D\(b-C*x0);
    if norm(x1-x0) <= toll;
        break
    end
    x0 = x1;
end
```

$x0 = (2, 1, -1)$

2.3.2.3.2 Algoritmo di Gauss-Seidel

ALG 2.5 (algoritmo di Gauss-Seidel). Metodo di risoluzione iterativa per un sistema lineare $Ax = b$ non singolare:

Algorithm 2.5 Algoritmo di Gauss-Seidel $[O \propto \rho(I - D^{-1}A)]$

1: $D = tril(A)$ // D è una matrice triangolare inferiore con i valori sotto la diagonale uguali ad A
2: $C = A - D$ // C è una matrice data dalla differenza tra A e D
3: **for** $(k = 1; k \leq kmax; k + 1)$ // Per $kmax$-volte
4: $x1 = D\backslash(b - Cx0)$ // Calcolo la k-esima approssimazione della soluzione del sistema
5: **if** $(norm(x1 - x0) \leq toll)$ // Se la differenza tra la soluzione precedente e quella corrente è minore della tolleranza
6: break // L'iterazione viene interrotta
7: **if end**
8: $x0 = x1$ // Memorizzo la soluzione corrente
9: **for end**

OSS 24. Costo computazionale: dipende dal raggio spettrale: minore è il raggio spettrale più rapida è la convergenza

MATLAB 16. Codice Matlab

```
A = [1 0 0; 2 3 0; 3 4 5];
b = [2; 7; 5];
kmax = 10;
toll = 0,01;
n = length(b);
x0 = zeros(n,1);
D = tril(A);
C = A-D;

for k = 1:kmax
    x1 = D\(b-C*x0);
    if norm(x1-x0) <= toll;
        break
    end
    x0 = x1;
end
```

$x0 = (2, 1, -1)$

Capitolo 3

Approssimazione di dati e funzioni

3.1 Definizioni

3.1.1 Approssimazione di una funzione

DEF 3.1 (approssimazione di una funzione)**.** Approssimare una funzione f significa sostituirla con una funzione $\tilde{f}$ simile a f ma con una forma più semplice:

$$\tilde{f} \approx f \tag{3.1}$$

3.1.2 Approssimazione di un insieme di dati

DEF 3.2 (approssimazione di un insieme di dati)**.** Approssimare un insieme di dati (x_i, y_i) significa determinare una funzione $\tilde{f}$ che abbia un andamento analogo a quello della funzione f che ha generato i dati

OSS 25 (scelta di $\tilde{f}$). Nell'approssimazione di una funzione f occorre:

- individuare una classe di funzioni approssimanti idonea a descrivere f
- adottare un criterio per la scelta di una particolare funzione approssimante $\tilde{f}$
- valutare la vicinanza di $\tilde{f}$ da f

3.1.3 Classi di funzioni approssimanti

OSS 26 (classi di $\tilde{f}$). Classi di funzioni approssimanti $\tilde{f}$ per funzioni $f \in C^k[a,b]$:

- polinomi algebrici di grado n:

$$\tilde{f} := p_n(x) = \sum_{k=0}^{n} a_k x^k \tag{3.2}$$

- polinomi trigonometrici di grado n e frequenza ω:

$$\tilde{f} := t_n(x) = a_0 + \sum_{k=1}^{n} \big(a_k \cos(k\omega x) + b_k \sin(k\omega x)\big) \tag{3.3}$$

- somme esponenziale di ordine n:

$$\tilde{f} := e_n(x) = \sum_{k=0}^{n} a_k e^{kx} \tag{3.4}$$

- funzioni spline di ordine n: funzioni polinomiali a tratti per cui:

$$\tilde{f} \in C^{n-1} \text{ a tratti} \tag{3.5}$$

3.2 Criteri per l'individuazione della funzione approssimante

3.2.1 Criterio dell'interpolazione

3.2.1.1 Definizione

DEF 3.3 (criterio dell'interpolazione polinomiale)**.** Se si considerano $n+1$ dati (x_i, y_i) con $y_i = f(x_i)$ bisogna determinare il polinomio di grado minimo passante per (x_i, y_i):

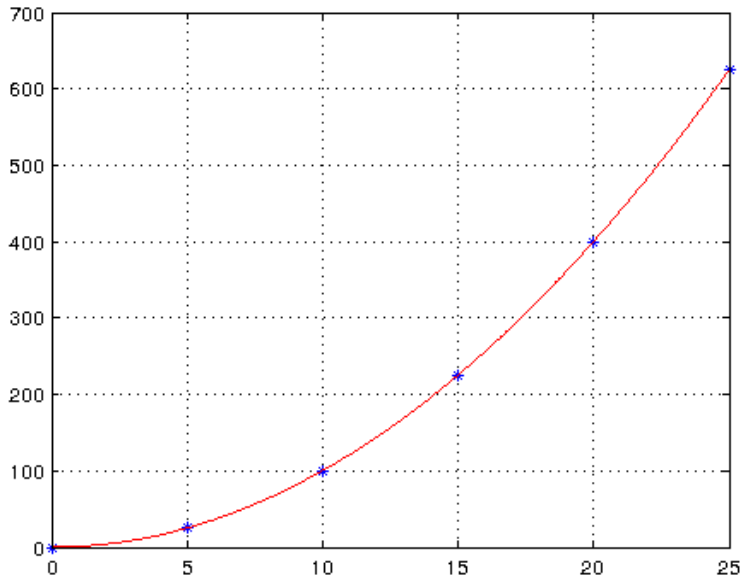
$$\tilde{f} \in p_n(x) = c_1 x^n + \cdots + c_n x + c_{n+1} \tag{3.6}$$

definito dai coefficienti $\{c_1, \ldots, c_{n+1}\}$

MATLAB 17. Codice Matlab

```
1  x = linspace(0,25,6);
2  f = inline('x.^2');
3  y = f(x);
4
5  n = length(x);
6  c = polyfit(x, y, n-1)
7  p = polyval(c, x)
8
9  x1 = linspace(0,25,1000);
10 y1 = f(x1);
11 plot(x,y,'*',x1,y1,'r');
12 grid on;
```

$c = (0, 0, 0, 1, 0, 0)$
$p = (0, 25, 100, 225, 400, 625)$



3.2.1.2 Unicità del polinomio interpolante

TEOR 3.2.1 (unicità del polinomio interpolante)**.** *Se* $x_i \neq x_j$ *allora* $\exists! p_n(x_i) = y_i$

Dimostrazione. Imponendo le condizioni di interpolazione si ottiene un sistema lineare di $n+1$ equazioni in n incognite:

$$\begin{cases} c_1 x_1^n + \cdots + c_n x_1 + c_{n+1} = y_1 \\ \quad \vdots \\ c_1 x_{n+1}^n + \cdots + c_n x_{n+1} + c_{n+1} = y_{n+1} \end{cases} \tag{3.7}$$

che si può riscrivere in forma matriciale come:

$$\begin{pmatrix} x_1^n & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{n+1}^n & \dots & x_{n+1} & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_{n+1} \end{pmatrix} \iff Vc = y \tag{3.8}$$

dove la matrice V è la matrice di Vandermonde; poiché gli x_i sono distinti:

$$det(V) = \prod_{i>j} x_i - x_j \tag{3.9}$$

la matrice V è non singolare e perciò:

$$\exists! c* = \begin{pmatrix} c_1^* \\ \vdots \\ c_{n+1}^* \end{pmatrix} \mid Vc^* = y \tag{3.10}$$

Il polinomio $p_n(x)$ i cui coefficienti sono soluzione del sistema $Vc = y$ è detto: polinomio interpolante i dati (x_i, y_i) oppure polinomio interpolante la funzione f nei punti x_i; i punti x_i sono chiamati nodi di interpolazione

□

3.2.1.3 Rappresentazione di Lagrange

PROP 3.2.2 (rappresentazione di Lagrange del polinomio di interpolazione)**.** *Dati i punti* $n+1$ *punti* $(x_i, f(x_i))$ *i polinomi fondamentali di Lagrange di grado* n *associati ai nodi* x_i *sono:*

$$I_j(x) = \prod_{\substack{i=1 \\ i \neq j}}^{n+1} \frac{(x - x_i)}{(x_j - x_i)} \tag{3.11}$$

dai quali si ricava il polinomio interpolante i punti $(x_i, f(x_i))$*:*

$$p_n(x) = \sum_{j=1}^{n+1} f(x_j) I_j(x) = \sum_{j=1}^{n+1} \left(f(x_j) \prod_{\substack{i=1 \\ i \neq j}}^{n+1} \frac{(x - x_i)}{(x_j - x_i)} \right) \tag{3.12}$$

che prende il nome di rappresentazione di Lagrange

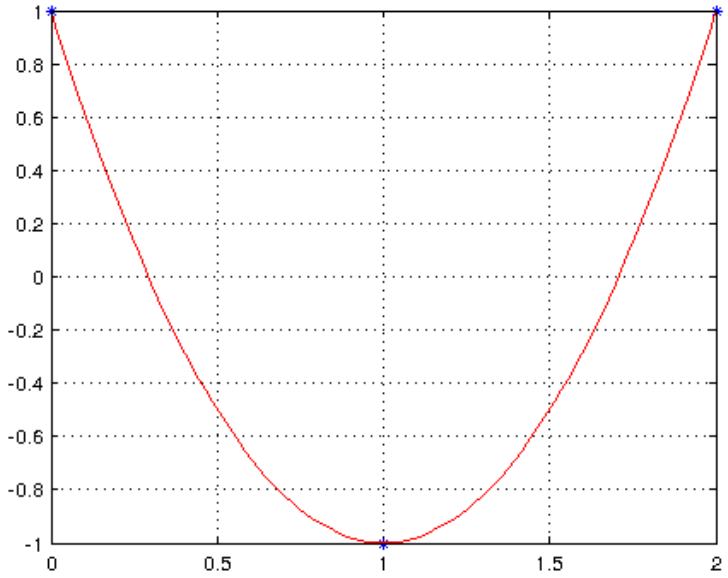
OSS 27. I polinomi fondamentali di Lagrange $I_j(x)$ dipendono dall'insieme completo dei nodi di interpolazione $\{(x_1, f(x_1)), \ldots, (x_{n+1}, f(x_{n+1}))\}$ pertanto, se si vuole aggiungere un nodo di interpolazione $(x_{n+2}, f(x_{n+2}))$ bisogna ricalcolare tutti i polinomi $I_j(x)$

e.g.: dati i punti $\{(0,1), (1,-1), (2,1)\}$ *i polinomi fondamentali di Lagrange sono:*

$$\begin{cases} I_1(x) = \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{1}{2}x^2 - \frac{3}{2}x + 1 \\ I_2(x) = \frac{(x-0)(x-2)}{(1-0)(1-2)} = x^2 - 2x \\ I_3(x) = \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{1}{2}x^2 - \frac{1}{2}x \end{cases}$$

da cui si ricava:

$$p_2(x) = 1I_1 - 1I_2 + 1I_3 = 2x^2 - 4x + 1$$



3.2.1.4 Rappresentazione di Newton

DEF 3.4 (differenza divisa di ordine n)**.** Dati $n+1$ punti $(x_i, f(x_i))$ si definisce differenza divisa di ordine n la quantità:

$$f(x_1, \dots, x_{n+1}) = \frac{f(x_2, \dots, x_{n+1}) - f(x_1, \dots, x_n)}{x_{n+1} - x_1} \tag{3.13}$$

OSS 28. Il rapporto incrementale è un caso particolare di differenza divisa di ordine 1:

$$f(x_1, x_2) = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{3.14}$$

PROP 3.2.3 (rappresentazione di Newton del polinomio di interpolazione)**.** *Dati n punti* $(x_i, f(x_i))$ *i polinomi:*

$$\begin{cases} 1 \\ x - x_1 \\ (x - x_1)(x - x_2) \\ \vdots \\ \prod_{i=1}^{n} (x - x_i) \end{cases} \tag{3.15}$$

sono detti polinomi fondamentali di Newton, dai quali si ricava la rappresentazione di Newton del polinomio interpolante i punti:

$$\begin{aligned} p_n(x) &= p_{n-1}(x) + f(x_1, \dots, x_n) \prod_{i=1}^{n} (x - x_i) = \\ &= f(x_1) + f(x_1, x_2)(x - x_1) + \cdots + f(x_1, \dots, x_n) \prod_{i=1}^{n} (x - x_i) = \\ &= \underbrace{\underbrace{\underbrace{f(x_1)}_{p_0(x)} + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)}_{p_1(x)} + \dots + \frac{f(x_2, \dots, x_{n+1}) - f(x_1, \dots, x_n)}{x_{n+1} - x_1}(x - x_1) \dots (x - x_n)}_{p_{n-1}(x)} \\ &\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{p_n(x)} \end{aligned} \tag{3.16}$$

OSS 29. Se si desidera aggiungere un nodo $(x_{n+1}, f(x_{n+1}))$ ai nodi di interpolazione è sufficiente aggiungere un addendo alla rappresentazione di Newton del polinomio di interpolazione

e.g.: dati i punti $\{(0,1), (1,-1), (2,1)\}$ *le differenze divise sono:*

$$\begin{array}{cc} x & f(x) \\ 0 & 1 \\ 1 & -1 \\ 2 & 1 \end{array} \qquad \begin{array}{c} \dfrac{f(x_b) - f(x_a)}{x_b - x_a} \\ \\ \dfrac{(-1) - 1}{1 - 0} = -2 \\ \\ \dfrac{1 - (-1)}{2 - 1} = 2 \end{array} \qquad \begin{array}{c} \dfrac{f(x_b, x_c) - f(x_a, x_b)}{x_c - x_a} \\ \\ \\ \\ \dfrac{2 - (-2)}{2 - 1} = 4 \end{array} \tag{3.17}$$

MATLAB 18. Codice Matlab
difdiv.m

```
function f = difdiv(x,y);
n = length(x);
for i = 1:n-1
    for j = n:-1:i+1
        y(j) = (y(j)-y(j-1))/(x(j)-x(j-i));
    end
end
f = y;
end
```

MATLAB 19. Codice Matlab
ex.m

```
x = [0 1 2];
y = [1 -1 1];

b = difdiv(x,y)
```

$b = (1, -2, 4)$

3.2.1.5 Algoritmo di Horner

ALG 3.1 (algoritmo di Horner). Algoritmo stabile e poco costoso che permette di valutare un polinomio in un punto z:

Algorithm 3.1 Algoritmo di Horner

1: $function\ p = interp(x, a, z)$ // definizione della funzione p con x vettore delle ascisse,
2: // a vettore delle differenze divise e z vettore dei punti da valutare
3: $n = length(z)$ // memorizza in n la lunghezza del vettore z
4: $p = a(n) * ones(1, length(z))$ // memorizza in p i valori dell'ultimo elemento di a
5: **for** $(k = n - 1; k \geq 1; k - 1)$ // per ogni differenza divisa (e valore di x corrispondente) a partire dall'ultima
6: $\quad p = p(z - x(k)) + a(k)$ // calcolo il polinomio di interpolazione nei punti z
7: **for end**

MATLAB 20. Codice Matlab
interp.m

```
function p = interp(x,a,z)
n = length(x);
p = a(n)*ones(1,length(z));

for k = n-1:-1:1
      p = p.*(z-x(k))+a(k);
end
end
```

MATLAB 21. Codice Matlab
ex.m

```
x = [0  1  2];
y = [1  -1  1];
a = difdiv(x,y);

z = x;
p = interp(x,a,z)
```

$p = (1, -1, 1)$

3.2.1.6 Errore di interpolazione

DEF 3.5 (errore di interpolazione)**.** Dato un polinomio $p_n(x)$ interpolante $f(x)$ nei punti $\{x_1, \ldots, x_n\}$ si definisce errore di interpolazione:

$$E_n(x) = f(x) - p_n(x) \tag{3.18}$$

dove per $x = x_i$ si ha $E_n(x_i) = f(x_i) - p_n(x_i) = 0$

3.2.1.7 Convergenza del polinomio di interpolazione

PROP 3.2.4 (convergenza del polinomio di interpolazione)**.** *Fissata una matrice di interpolazione contenente i nodi:*

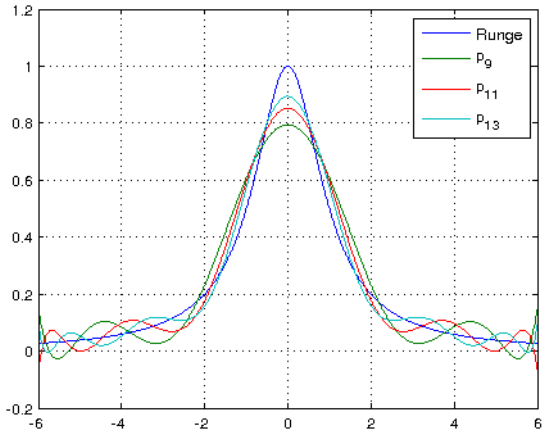
$$\begin{matrix} x_1 & & & \\ x_1 & x_2 & & \\ \vdots & & & \\ x_1 & x_2 & \dots & x_n \end{matrix} \tag{3.19}$$

è sempre possibile determinare una funzione f per la quale:

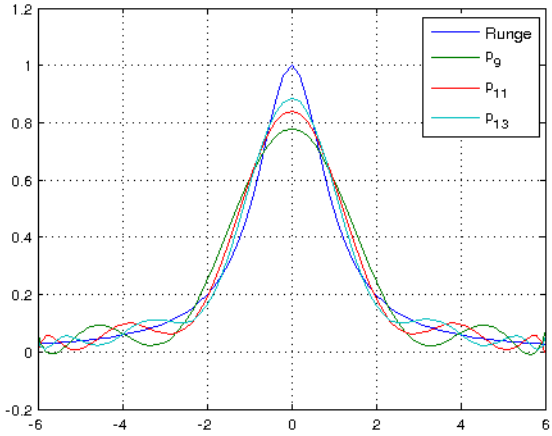
$$\lim_{n\to\infty} ||E_n||_\infty \neq 0 \tag{3.20}$$

La convergenza dipende strettamente dalla scelta dei nodi di interpolazione e dalla regolarità di f

OSS 30. Se i nodi sono scelti equispaziati in $[a, b]$ la condizione $f \in C^\infty[a, b]$ non garantisce la convergenza di f
e.g.: dati x_i nodi equispaziati nell'intervallo $[-5, 5]$ e $f(x) = \frac{1}{1+x^2}$ (funzione di Runge) allora $\lim_{n\to\infty} ||E_n||_\infty = \infty$ Se invece i nodi di interpolazione sono scelti in corrispondenza degli zeri dei



polinomi di Chebyshev allora la condizione $f \in C^1$ garantisce la convergenza
e.g.: dati $x_i = \frac{b-a}{2} z_i + \frac{b+a}{2}$ nell'intervallo $[a = -5, b = 5]$ allora $\lim_{n\to\infty} ||E_n||_\infty = 0$



3.2.1.8 Funzione polinomiale a tratti (spline)

DEF 3.6 (funzione polinomiale a tratti di grado d)**.** Funzione rappresentata in $[a, b]$ da un'unione di tratti contigui di polinomi algebrici diversi di grado d

OSS 31. Le funzioni polinomiali a tratti sono continue ($f \in C^0$) ma generalmente non derivabili nei punti di raccordo

OSS 32. Generalmente le funzioni polinomiali a tratti vengono utilizzate quando il comportamento oscillatorio dei polinomi di interpolazione cresce in modo non trascurabile al crescere del grado del polinomio

DEF 3.7 (spline cubica)**.** Dati $n+1$ punti $(x_i, f(x_i))$ di ascissa differente, una spline cubica $S_3(x)$ è una funzione polinomiale a tratti che soddisfa le seguenti condizioni:

- ha forma: $S_3(x) = a_i + b_i x + c_i x^2 + d_i x^3 \ \forall x \in [x_1, x_{n+1}]$ ($4n$ incognite: a_i, b_i, c_i, d_i)
- la derivata k-esima sinistra coincide con la derivata k-esima destra in ogni punto: $S_3^{(k)}(x_i^-) = S_3^{(k)}(x_i^+)$ ($n-1$ equazioni)
- interpola la funzione: $S_3(x_i) = f(x_i)$ ($n+1$ equazioni)

Con queste condizioni si ricava un sistema lineare di

$$3(n-1) + (n+1) = 4n - 2 \tag{3.21}$$

equazioni in $4n$ incognite; per definire univocamente una spline si può scegliere tra le seguenti condizioni aggiuntive:

- spline cubiche naturali: $S_3^2(x_1) = 0$ e $S_3^2(x_{n+1}) = 0$
- condizioni *not-a-knot*: $S_3^2(x_2^-) = S_3^2(x_2^+)$ e $S_3^2(x_n^-) = S_3^2(x_n^+)$
- $S_3^1(x_1) = f'(x_1)$ e $S_3^1(x_{n+1}) = f'(x_{n+1})$

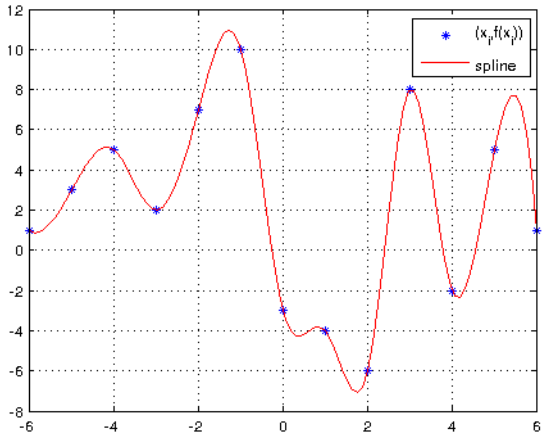
OSS 33. Se si aggiunge una delle condizioni aggiuntive si può determinare in modo univoco una spline risolvendo un sistema lineare in $n+1$ incognite $M_i = S_3^2(x_i)$ simmetrico, a diagonale dominante e tridiagonale

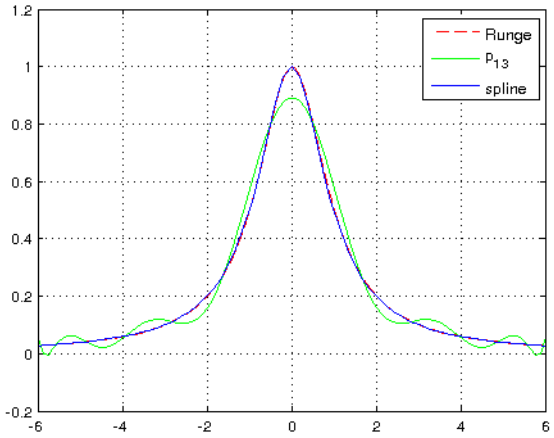
MATLAB 22. Codice Matlab

```
x = linspace(-6,6,13);
y = [1  3  5  2  7  10  -3  -4  -6  8  -2  5  1]
z = linspace(-6,6);
s = spline(x,y,z);

plot(x,y,'*',z,s,'r')
legend('(x_i,f(x_i))','spline')
grid on
```

MATLAB 23. Codice Matlab

```
x = linspace(-6,6);
f = inline('1./(1+x.^2)');
y = f(x);

c1 = polyfit(x,y,13);
y1 = polyval(c3,x);
z = linspace(-6,6,65);
s = spline(x,y,z);

plot(x,y,x,y1,z,s)
legend('Runge','p_{13}','spline')
grid on
```





3.2.1.9 Convergenza delle funzioni polinomiali a tratti

PROP 3.2.5 (convergenza delle spline cubiche)**.** *Data una spline cubica S_3 interpolante i punti $(x_i, f(x_i))$, $h = max_{1\leq i\leq n}(x_{i+1} - x_i)$ e $f \in C^4$ in $[a, b]$ allora per $h \to 0$ e $p = 0, 1, 2, 3$:*

$$||f^{(p)} - S_3^{(p)}||_\infty = O(h^{4-p}) \tag{3.22}$$

cioè risulta che S_3 e le sue derivate fino a quelle di ordine 3 convergono uniformemente a f e alle sue derivate

OSS 34. Per questo motivo è sufficiente conoscere f per approssimare le sue derivate

OSS 35. $O(h^4)$ è il massimo ordine di convergenza che si può ottenere con le spline cubiche:

$$||f - S_3||_\infty = O(h^4) \tag{3.23}$$

anche quando $f \in C^k$ con $k > 4$

3.2.2 Criterio dei minimi quadrati

3.2.2.1 Definizione

DEF 3.8 (criterio dei minimi quadrati)**.** Dati m punti $(x_i, f(x_i))$ e le funzioni base $\phi_k(x)$ con $1 \leq k \leq n$ la funzione approssimante $\tilde{f}$ è una combinazione lineare a coefficienti costanti di $\phi_k(x)$:

$$\tilde{f}(x_i) = \sum_{k=1}^{n} c_k \phi_k(x_i) \tag{3.24}$$

e se il residuo:

$$\epsilon^2(c_1, \ldots, c_n) = \sum_{i=1}^{m} (f(x_i) - \tilde{f}(x_i))^2 \tag{3.25}$$

è minimo allora $\tilde{f}$ viene definita migliore approssimazione dei dati $(x_i, f(x_i))$ secondo il criterio dei minimi quadrati

OSS 36. Il criterio dei minimi quadrati viene adottato per approssimare al meglio un insieme di dati, per generare valutazioni in punti che giacciono al di fuori dell'intervallo di interpolazione o per costruire funzioni approssimanti dipendenti da un numero di nodi molto minore del numero di dati: $n \ll m$

3.2.2.2 Determinazione della migliore approssimazione di f

PROP 3.2.6 (determinazione della migliore approssimazione di f secondo il criterio dei minimi quadrati)**.** *Gli $n \ll m$ coefficienti c_k si ricavano imponendo la condizione necessaria per l'esistenza del minimo (per $1 \leq j \leq n$):*

$$\frac{\partial \epsilon^2}{\partial c_j} = 0 \to 2\sum_{i=1}^{m} \left(f(x_i) - \sum_{k=1}^{n} c_k \phi_k(x_i) \right) \phi_j(x_i) = 0 \to \sum_{k=1}^{n} c_k \left(\sum_{i=1}^{m} \phi_k(x_i)\phi_j(x_i) \right) = \sum_{i=1}^{m} f(x_i)\phi_j(x_i) \tag{3.26}$$

da cui si ricava il sistema di equazioni lineari $Bc = d$ con $B \in \mathbb{R}^{n \times n}$, $c \in \mathbb{R}^n$ e $d \in \mathbb{R}^n$:

$$\begin{pmatrix} \sum_{i=1}^{m} \phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{m} \phi_2(x_i)\phi_1(x_i) & \ldots & \sum_{i=1}^{m} \phi_n(x_i)\phi_1(x_i) \\ \sum_{i=1}^{m} \phi_1(x_i)\phi_2(x_i) & \sum_{i=1}^{m} \phi_2(x_i)\phi_2(x_i) & \ldots & \sum_{i=1}^{m} \phi_n(x_i)\phi_2(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} \phi_1(x_i)\phi_n(x_i) & \sum_{i=1}^{m} \phi_2(x_i)\phi_n(x_i) & \ldots & \sum_{i=1}^{m} \phi_n(x_i)\phi_n(x_i) \end{pmatrix} \begin{pmatrix} c1 \\ c2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} f(x_i)\phi_1(x_i) \\ \sum_{i=1}^{m} f(x_i)\phi_2(x_i) \\ \vdots \\ \sum_{i=1}^{m} f(x_i)\phi_n(x_i) \end{pmatrix} \tag{3.27}$$

detto sistema delle equazioni normali;
posto:

$$A = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_2) & \ldots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \ldots & \phi_n(x_m) \end{pmatrix} \tag{3.28}$$

si ha:

$$B = A^T A \tag{3.29}$$

da cui si dimostra che B è una matrice simmetrica semidefinita positiva (cioè $x^T Bx \geq 0 \forall x \neq \vec{o}$);
B è definita positiva se e solo se le colonne di A sono linearmente indipendenti; se tale condizione è soddisfatta il sistema ammette una sola soluzione e tale soluzione rende minimo il residuo ϵ^2;
se le colonne di A sono linearmente dipendenti il sistema ammette infinite soluzioni, ma solo una di

queste ha norma 2 minima: pertanto si assume come soluzione il vettore per cui:

$$\begin{aligned} &\min\left\{\sum_{i=1}^{m}\left(f(x_i)-(c_1^*\phi_1(x_i)+c_2^*\phi_2(x_i)+\cdots+c_n^*\phi_n(x_i))\right)^2\right\} \\ &\min\left\{||c^*||_2^2\right\} \end{aligned} \tag{3.30}$$

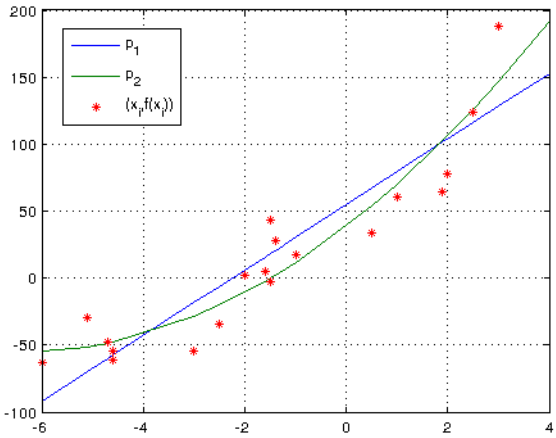
e.g.: dati i punti (x_i, y_i) per determinare i coefficienti del polinomio di grado 1 $p_1(x) = c_1x + c_2$ si risolve il sistema:

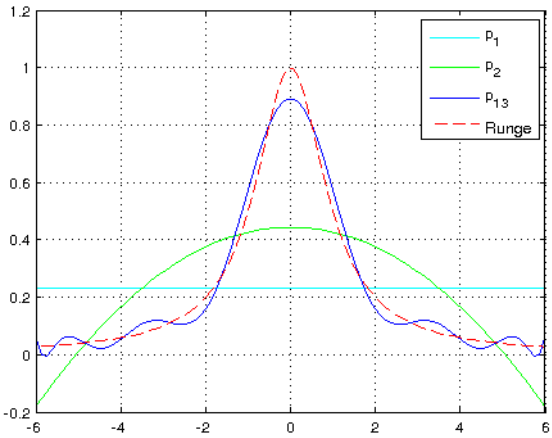
$$\begin{pmatrix} \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & m \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} f(x_i)x_i \\ \sum_{i=1}^{m} f(x_i) \end{pmatrix} \tag{3.31}$$

dove $p_1(x)$ prende il nome di retta di regressione lineare

MATLAB 24. Codice Matlab

```
x = [-6 -5.1 -4.7 -4.6 -4.6 -3 -2.5 -2 -1.6 -1.5 -1.5 -1.4 -1 0.5 ...
     1 1.9 2 2.5 3 4];
y = [-63 -30 -48 -61 -54 -54 -34 2 5 43 -3 28 17 34 61 64 78 124 ...
     188 200];

c1 = polyfit(x,y,1);
y1 = polyval(c1,x);
c2 = polyfit(x,y,2);
y2 = polyval(c2,x);

plot(x,y1,x,y2,x,y,'*')
legend('p_1','p_2','(x_i,f(x_i))')
grid on
```





Capitolo 4

Equazioni non lineari

4.1 Definizioni

4.1.1 Risolvere un'equazione non lineare

PROP 4.1.1 (risolvere un'equazione non lineare)**.** *La ricerca di equazioni non lineari consiste nel trovare i punti in cui:*

$$f(x) = 0 \tag{4.1}$$

OSS 37. I metodi numerici sono utili quando:

- non è possibile trovare una soluzione analitica
- trovare una soluzione analitica risulta lungo o complicato
- la funzione è assegnata mediante valori numerici (e non mediante un'espressione analitica)

OSS 38. I metodi numerici per il calcolo delle radici di un'equazione non lineare sono di tipo iterativo: a partire da un certo valore x_0 si determina una successione di valori $\{x_n\}$ che, sotto opportune condizioni, converge ad una radice α dell'equazione assegnata

4.1.2 Ordine di convergenza di una successione

DEF 4.1 (ordine di convergenza di una successione x_n)**.** Si definisce ordine di convergenza di una successione x_n convergente ad α il numero:

$$p \geq 1 \in \mathbb{R} \mid \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|^p} = C \neq \begin{cases} 0 \\ \infty \end{cases} \tag{4.2}$$

da cui si ricava che:

$$\begin{cases} p = 1 \implies C < 1 \text{ convergenza lineare} \\ 1 < p < 2 \implies \text{convergenza superlineare} \\ p = 2 \implies \text{convergenza quadratica} \\ p = 3 \implies \text{convergenza cubica} \end{cases} \tag{4.3}$$

OSS 39. Se un metodo ha ordine di convergenza p allora per n sufficientemente grande:

$$|\alpha - x_{n+1}| \approx C|\alpha - x_n|^p \tag{4.4}$$

e quindi:

$$|\alpha - x_n| \leq 10^{-k} \implies |\alpha - x_{n+1}| \approx C10^{-kp} \tag{4.5}$$

cioè se x_n ha k decimali corretti, il numero di decimali corretti in x_{n+1} è dell'ordine di kp

OSS 40. Se x_0 è "sufficientemente vicino" ad α e f è "sufficientemente regolare" allora i metodi numerici convergono alle radici di f

4.2 Algoritmi di risoluzione di equazioni non lineari

4.2.1 Algoritmo di bisezione

DEF 4.2 (algoritmo di bisezione)**.** Se $f \in C([a,b])$ e $f(a)f(b) < 0$ allora esiste almeno un punto $\alpha \in (a,b)$ tale che $f(\alpha) = 0$ e quindi si procede per successivi dimezzamenti dell'intervallo contenente α

OSS 41. Ordine di convergenza: $p = 1$ (molto lento)

OSS 42. A causa della lentezza della sua convergenza il metodo di bisezione viene utilizzato generalmente per inizializzare metodi più rapidamente convergenti

4.2.2 Algoritmo delle secanti

ALG 4.1 (algoritmo delle secanti). Metodo di risoluzione di un'equazione non lineare:

Algorithm 4.1 Algoritmo delle secanti

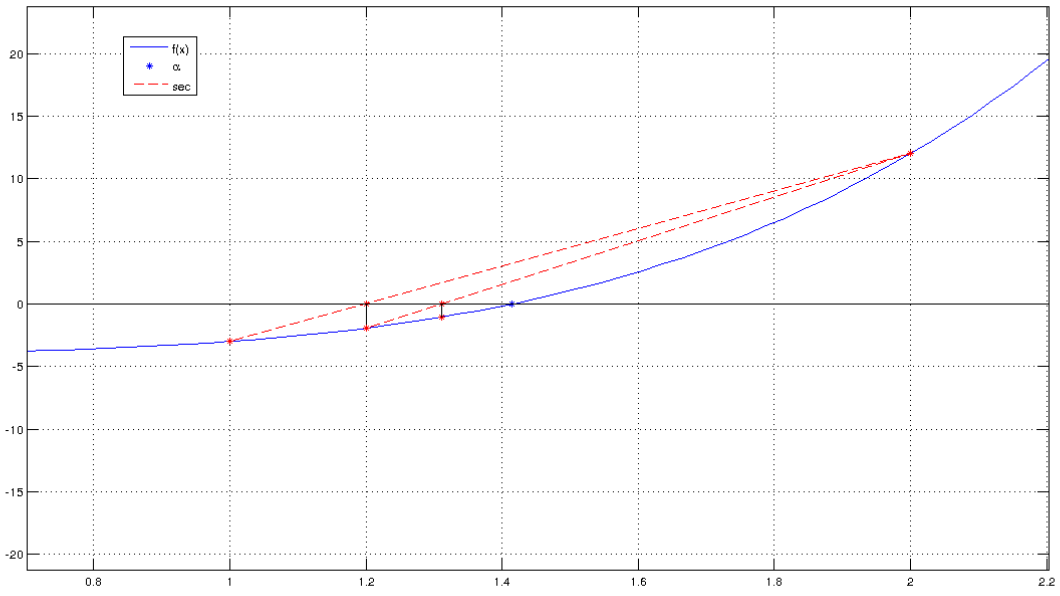
1: $f = inline('\ldots')$ // f rappresenta l'equazione da risolvere
2: $x0 = \ldots$ // estremo sinistro dell'intervallo in cui cercare α
3: $x1 = \ldots$ // estremo destro dell'intervallo in cui cercare α
4: $nmax = \ldots$ // massimo numero di iterazioni
5: $toll = \ldots$ // tolleranza minima sotto la quale si interrompe il ciclo di iterazione
6: **for** $(n = 1; n \leq nmax; n+1)$ // da 1 a $nmax$ volte
7: $x2 = x1 - f(x1) * (x1 - x0)/(f(x1) - f(x0))$ // $x_{n+1} = x2$ viene definito come l'intersezione tra
8: // l'asse delle ascisse della retta passante per i punti $(x_{n-1}, f(x_{n-1}))$ e $(x_n, f(x_n))$
9: **if** $abs(x2 - x1) \leq toll$ // se la differenza tra $|x_{n+1} - x_n|$ è inferiore alla tolleranza
10: $break$ // il ciclo viene interrotto
11: **if end**
12: $x0 = x1$ // l'estremo sinistro dell'intervallo diventa x_n
13: $x1 = x2$ // l'estremo destro dell'intervallo diventa x_{n+1}
14: **for end**
15: $x2$ // x_2 è l'approssimazione della radice dell'equazione f

MATLAB 25. Codice Matlab

```
f = inline('x.^4-4');
x0 = 1;
xa = x0;
x1 = 2;
xb = x1;
nmax = 100;
toll = 1.0*10^(-10);
z = zeros(1,nmax);
for n = 1:nmax
    x2 = x1-f(x1)*(x1-x0)/(f(x1)-f(x0));
    if abs(x2-x1) <= toll;
        break
    end
    z(n) = x0;
    x0 = x1;
    x1 = x2;
end
```

```
18 z(n) = x2;
19
20 x = [z(1) z(2)];
21 y = f(x);
22 xa = [z(3) z(3)];
23 ya = [0 f(z(3))];
24 x1 = [z(3) z(2)];
25 y1 = f(x1);
26 xb = [z(4) z(4)];
27 yb = [0 f(z(4))];
28 x2 = linspace(0,3);
29 y2 = f(x2);
30 y3 = linspace(0,0);
31 y4 = f(z(n));
32
33 plot(x2,y2,'b',z(n),y4,'*', ...
34      x,y,'r--',x,y,'r*', ...
35      xa,ya,'r*',xa,ya,'black', ...
36      x1,y1,'r--',x1,y1,'r*', ...
37      xb,yb,'r*',xb,yb,'black', ...
38      x2,y3,'black')
39 legend('f(x)','\alpha','sec')
40 grid on
```

$x0 = 2.000000000000000$
$x0 = \mathbf{1}.200000000000000$
$x0 = \mathbf{1}.310661764705882$
$x0 = \mathbf{1.4}42979105271929$
$x0 = \mathbf{1.41}0915551310193$
$x0 = \mathbf{1.414}114438843506$
$x0 = \mathbf{1.41421}3909809020$
$x0 = \mathbf{1.414213562}336565$



OSS 43. Se tutte le successive approssimazioni di α sono comprese in $[a, b]$, se α è una radice semplice di f, se $f \in C^2([a, b])$ e se le distanze $|\alpha - x_0|$ e $|\alpha - x_1|$ sono sufficientemente piccole allora:

$$\lim_{n\to\infty} x_n = \alpha \tag{4.6}$$

OSS 44. Se α è semplice l'ordine di convergenza è: $p = \frac{1+\sqrt{5}}{2} \approx 1.618$

4.2.3 Algoritmo delle tangenti

ALG 4.2 (algoritmo delle tangenti o di Newton-Raphson). Metodo di risoluzione di un'equazione non lineare:

Algorithm 4.2 Algoritmo delle tangenti

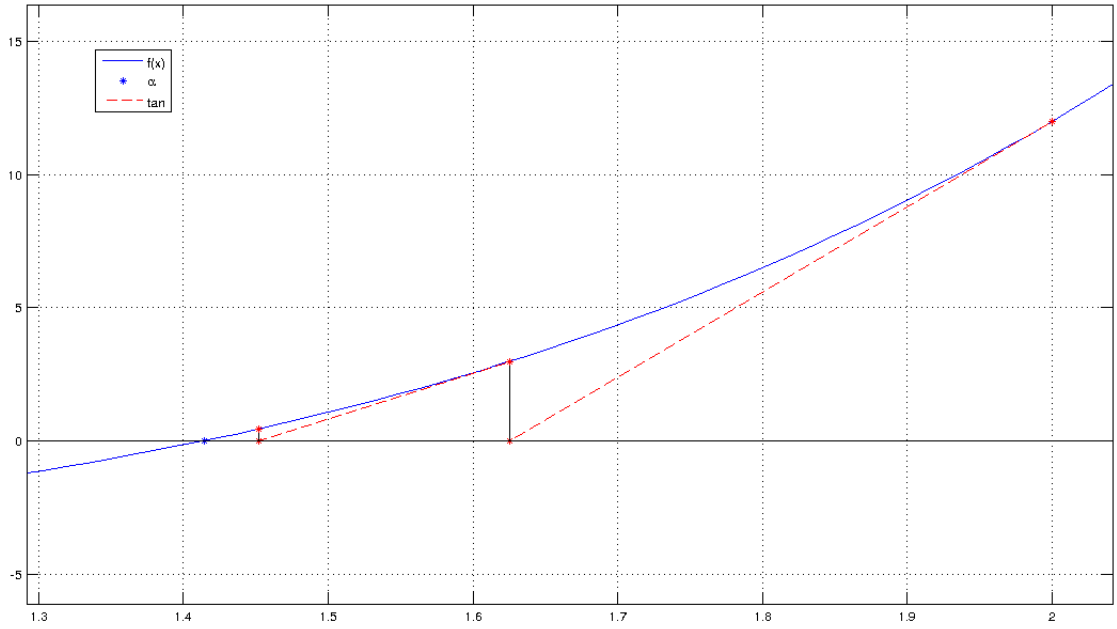
1: $f = inline('...')$ // f rappresenta l'equazione da risolvere
2: $fd = inline('...')$ // f rappresenta l'equazione da risolvere
3: $x0 = \ldots$ // estremo sinistro dell'intervallo in cui cercare α
4: $nmax = \ldots$ // massimo numero di iterazioni
5: $toll = \ldots$ // tolleranza minima sotto la quale si interrompe il ciclo di iterazione
6: **for** $(n = 1; n \leq nmax; n + 1)$ // da 1 a $nmax$ volte
7: $x1 = x0 - f(x0)/fd(x0)$ // $x_{n+1} = x2$ viene definito come l'intersezione tra
8: // l'asse delle ascisse della retta passante per i punti $(x_{n-1}, f(x_{n-1}))$ e $(x_n, f(x_n))$
9: **if** $abs(x1 - x0) \leq toll$ // se la differenza tra $|x_{n+1} - x_n|$ è inferiore alla tolleranza
10: $break$ // il ciclo viene interrotto
11: **if end**
12: $x0 = x1$ // l'estremo sinistro dell'intervallo diventa x_n
13: **for end**
14: $x1$ // x_1 è l'approssimazione della radice dell'equazione f

MATLAB 26. Codice Matlab

```
f = inline('x.^4-4');
fd = inline('4*x.^3');
x0 = 2;
nmax = 100;
toll = 1.0*10^(-10);
z = zeros(1,nmax);
for i = 1:nmax
    x1 = x0-f(x0)/fd(x0);
    if abs(x1-x0) <= toll;
        break
    end
    z(i) = x0;
    x0 = x1;
end
z(i) = x0;

x = linspace(1,2.5);
y = f(x);
xa = [z(1) z(2)];
ya = [f(z(1)) 0];
xb = [z(2) z(3)];
yb = [f(z(2)) 0];
xx = linspace(1,2.5);
yy = zeros(1,100);


plot(x,y,'b', ...
    z(i),f(z(i)),'b*', ...
    xa,ya,'r--',xa,ya,'r*', ...
    [z(2) z(2)],[f(z(2)) 0],'black', ...
```

```
31        xb,yb,'r--',xb,yb,'r*',  ...
32        [z(3)  z(3)],[f(z(3))  0],'black',  ...
33        z(3),f(z(3)),'r*',  ...
34        xx,yy,'black')
35  legend('f(x)','\alpha','tan')
36  grid  on
```

$x0 = 2.000000000000000$
$x0 = \mathbf{1}.625000000000000$
$x0 = \mathbf{1.4}51795061447428$
$x0 = \mathbf{1.41}5647807027460$
$x0 = \mathbf{1.41421}5740529996$
$x0 = \mathbf{1.4142135623}78127$



OSS 45. Se tutte le successive approssimazioni di α sono comprese in $[a, b]$, se α è una radice semplice di f, se $f \in C^2([a, b])$ e se la distanza $|\alpha - x_0|$ è sufficientemente piccola allora:

$$\lim_{n\to\infty} x_n = \alpha \tag{4.7}$$

OSS 46. L'ordine di convergenza è: $p = 2$ se α è semplice; $p = 1$ se α non è semplice

4.2.4 Algoritmo del punto fisso

ALG 4.3 (algoritmo del punto fisso). Metodo di risoluzione di un'equazione non lineare in cui il problema del calcolo delle radici di $f(x)$ (cioè i valori α per cui $f(\alpha) \equiv 0$) viene riformulato nel problema equivalente del calcolo delle rafici di $x - g(x)$ (cioè i valori α per cui $\alpha - g(\alpha) \equiv 0$) dove ciascun valore α si definisce punto fisso di $g(x)$;
la funzione $g(x)$ viene definita come:

$$f(\alpha) \equiv 0 \implies \alpha + \frac{f(\alpha)}{k} \equiv g(\alpha) \equiv \alpha \implies g(x) = x + \frac{f(x)}{k} \tag{4.8}$$

lo scopo è determinare α in modo che $g(\alpha) \equiv \alpha$ a partire da x_0 e come limite della successione:

$$x_{n+1} = g(x_n) \tag{4.9}$$

Algorithm 4.3 Algoritmo del punto fisso

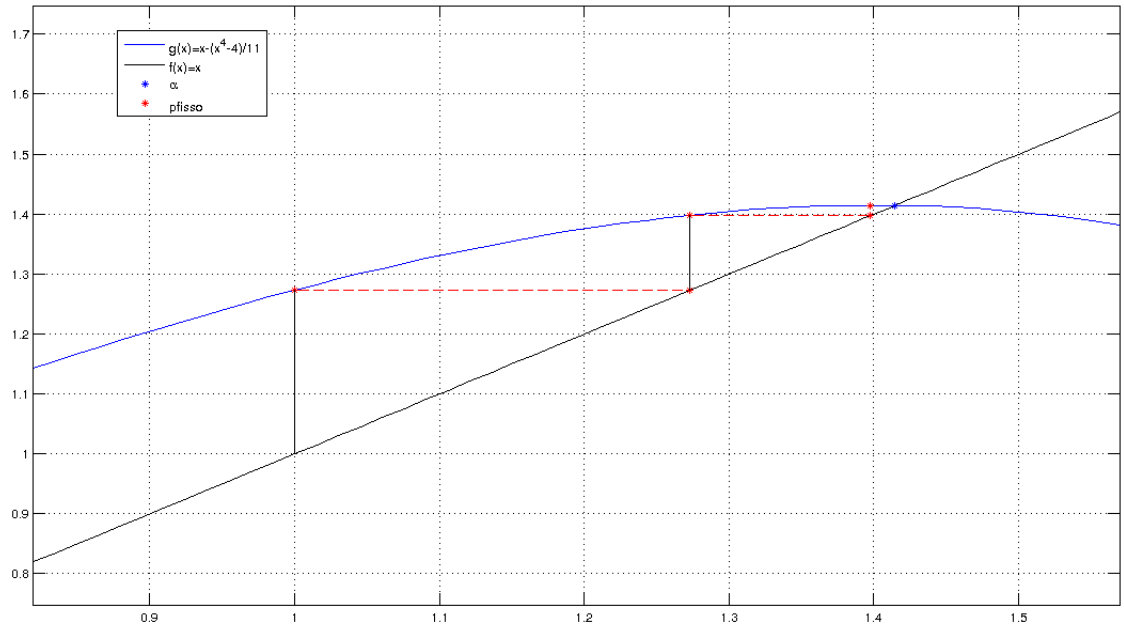
```
1:  g = inline('...')          // g rappresenta la funzione di cui si vogliono trovare i punti fissi
2:  x0 = ...                   // estremo sinistro dell'intervallo in cui cercare α
3:  nmax = ...                 // massimo numero di iterazioni
4:  toll = ...                 // tolleranza minima sotto la quale si interrompe il ciclo di iterazione
5:  for (n = 1; n ≤ nmax; n + 1)   // da 1 a nmax volte
6:      x1 = g(x0)             // x_{n+1} viene definito come il valore che assume g(x) in x_n
7:      if abs(x1 − x0) ≤ toll // se la differenza tra |x_{n+1} − x_n| è inferiore alla tolleranza
8:          break              // il ciclo viene interrotto
9:      if end
10:     x0 = x1                // l'estremo sinistro dell'intervallo diventa x_n
11: for end
12: x1                         // x_1 è l'approssimazione della radice dell'equazione f
```

MATLAB 27. Codice Matlab

```
1  g = inline('x-(x.^4-4)/11');
2  x0 = 1;
3  nmax = 100;
4  toll = 1.0*10^(-10);
5  z = zeros(1,nmax);
6  for i = 1:nmax
7      x1 = g(x0);
8      if abs(x1-x0) <= toll;
9          break
10     end
11     z(i) = x0;
12     x0 = x1;
13 end
14 z(i) = x0;
15
16 f = inline('x');
17 x = linspace(0.5,2);
18 y = g(x);
19 yx = f(x);
20 xa = [z(1) z(2)];
21 ya = [g(z(1)) f(z(2))];
22 xb = [z(2) z(3)];
```

```
yb = [g(z(2)) f(z(3))];
xx = linspace(0.5,2);
yy = zeros(1,100);

plot(x,y,'b', ...
     x,yx,'black', ...
     z(i),g(z(i)),'b*', ...
     xa,ya,'r*',xa,ya,'r--', ...
     [z(1) z(1)],[f(z(1)) g(z(1))],'black', ...
     xb,yb,'r--',xb,yb,'r*', ...
     [z(2) z(2)],[f(z(2)) g(z(2))],'black', ...
     z(3),g(z(3)),'r*', ...
     xx,yy,'black')
legend('g(x)=x-(x^4-4)/11','f(x)=x','\alpha','pfisso')
grid on
```

$x0 = \mathbf{1}.000000000000000$
$x0 = \mathbf{1}.272727272727273$
$x0 = \mathbf{1}.397830500897231$
$x0 = \mathbf{1.414}390239825487$
$x0 = \mathbf{1.4142}08489661399$
$x0 = \mathbf{1.414213}707013457$
$x0 = \mathbf{1.4142135}58248080$
$x0 = \mathbf{1.41421356}2490736$
$x0 = \mathbf{1.414213562}369740$



OSS 47. Se $k = -f'(x)$ allora $g(x) = x - \frac{f(x)}{f'(x)}$ si ottiene il metodo delle tangenti

OSS 48. Se α è un punto fisso di g, se $g \in C^1(I)$ con $\alpha \in I$, se $|g'(\alpha)| < 1$ e se $|\alpha - x_0|$ è sufficientemente piccolo allora la successione $x_n = g(x_{n-1})$ ha $\lim_{n\to\infty} x_n = \alpha$

OSS 49. Se esiste un intorno I di α in cui $|g'(x)| > 1 \forall x \in I$ allora $\lim_{n\to\infty} x_n \neq \alpha$

TEOR 4.2.1. *Sia I un intervallo dell'asse reale. Se:*

- $g(x) \in I \quad \forall x \in I$
- $g \in C^1(I)$
- $\exists m < 1 \mid \quad |g'(x)| \leq m \forall x \in I$ *allora esiste uno e un solo punto fisso α di g in I e per* $x_{n+1} = g(x_n)$

si ha:

$$\lim_{n\to\infty} x_n = \alpha \quad \forall x_0 \in I \tag{4.10}$$

e

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{\alpha - x_n} = g'(\alpha) \tag{4.11}$$

4.3 Comandi Matlab per il calcolo delle radici

4.3.1 *fzero* e *roots*

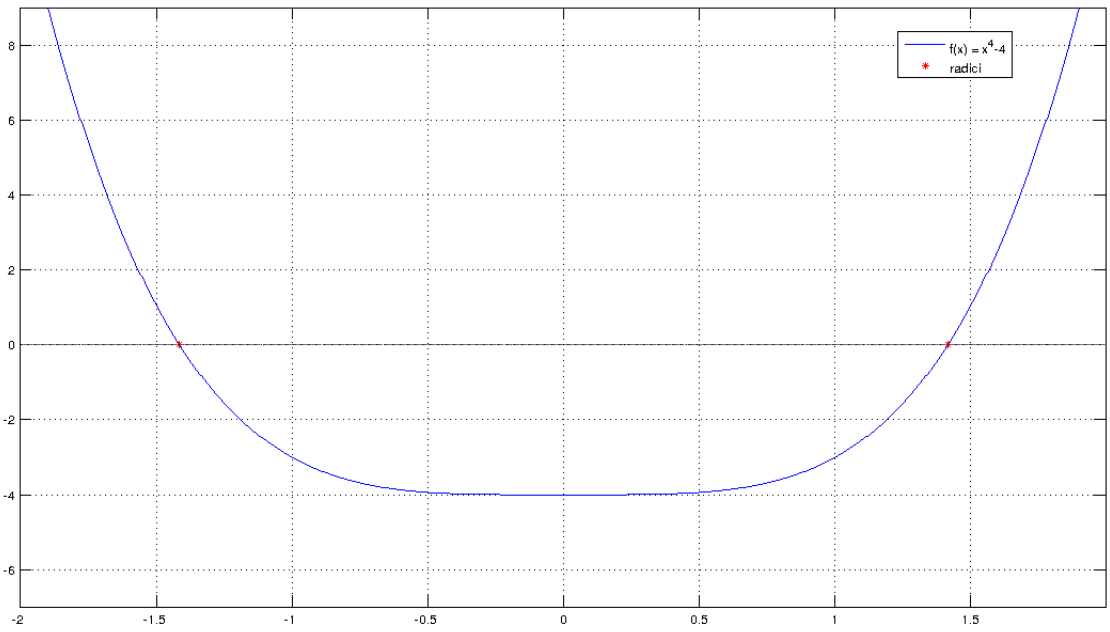
Comandi per ottenere le radici di un'equazione a partire dalla sua espressione analitica o dai suoi coefficienti

MATLAB 28. Codice Matlab

```
f = inline('x.^4-4');
c = [1 0 0 0 -4];
x0 = 1;
toll = 1.0*10^(-10);

r1 = fzero(f,x0,toll); % tramite l'espressione analitica
r2 = roots(c); % tramite i coefficienti

x = linspace(-2,2,1000);
y = f(x);
g = inline('0');
y1 = g(x);

plot(x,y, ...
     r2(1),0,'r*',r2(4),0,'r*', ...
     x,y1)
legend('f(x) = x^4-4','radici')
grid on
```

$r1 = \mathbf{1.414213562373095}$

$$x2 = \begin{pmatrix} \mathbf{-1.414213562373097 + 0.000000000000000}i \\ \mathbf{0.000000000000000 + 1.414213562373096}i \\ \mathbf{0.000000000000000 - 1.414213562373096}i \\ \mathbf{1.414213562373096 + 0.000000000000000}i \end{pmatrix}$$



Capitolo 5

Integrali definiti

5.1 Formule di quadratura o di integrazione numerica

OSS 50. I metodi numerici per il calcolo dell'integrale definito $\int_a^b f(x)dx$ di una funzione $f(x)$ sono necessari quando:

- non è possibile calcolare analiticamente l'integrale
- è complicato o lungo trovare il valore dell'integrale con metodi analitici
- f è nota solo per punti
- f è valutabile per ogni valore di x tramite un algoritmo

DEF 5.1 (formula di quadratura o formula di integrazione numerica)**.** Formula che permette di valutare da un punto di vista numerico (approssimato) il valore di un integrale:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{5.1}$$

dove i punti x_i prendono il nome di nodi di quadratura e i numeri w_i coefficienti (o pesi) di quadratura

5.2 Formule di quadratura interpolatorie di Newton-Cotes

5.2.1 Definizione

DEF 5.2 (formule di quadratura interpolatorie di Newton-Cotes)**.** Dati n nodi di quadratura $x_1, \ldots, x_n$ nell'intervallo $[a, b]$ le formule di quadratura interpolatorie si ottengono approssimando l'integrale assegnato con l'integrale del polinomio $p_{n-1}(f; x)$ che interpola la funzione f nei nodi di quadratura:

$$\int_a^b f(x)dx \approx \int_a^b p_{n-1}(f;x)dx = \int_a^b \sum_{i=1}^{n} l_i(x) f(x_i)dx = \sum_{i=1}^{n} \left(\int_a^b l_i(x)dx \right) f(x_i) = \sum_{i=1}^{n} w_i f(x_i) \tag{5.2}$$

dove l_i sono i polinomi fondamentali di Lagrange

5.2.2 Formula del rettangolo

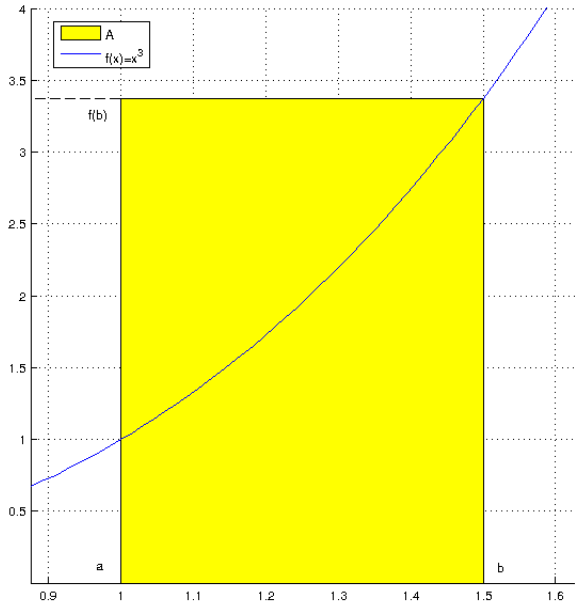
DEF 5.3 (formula del rettangolo)**.** Per $n = 1$ e $x_1 = a$:

$$\int_a^b f(x)dx \approx \int_a^b p_0(f;x)dx = \int_a^b f(b)dx = f(b)(b-a) \tag{5.3}$$

MATLAB 29. Codice Matlab

```
f = inline('x.^3');
x = linspace(0.5,2);
y = f(x);

a = 1;
b = 1.5;
I = f(a)*(b-a)

patch([a a b b a],[0 f(b) f(b) 0 0],'y');
hold on
plot(x,y,[0.5 1],[f(b) f(b)],'black--');
legend('A','f(x)=x^3')
grid on
```

$I = \mathbf{1}.687500000000000$



Per $n = 1$ e $x_1 = \frac{a+b}{2}$ (formula del punto medio):

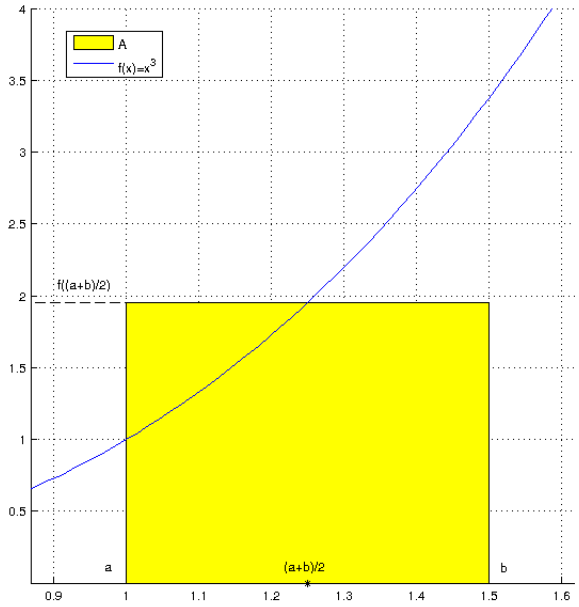
$$\int_a^b f(x)dx \approx f\left(\frac{a+b}{2}\right)(b-a) \tag{5.4}$$

MATLAB 30. Codice Matlab

```
f = inline('x.^3');
x = linspace(0.5,2);
y = f(x);

```

```
a = 1;
b = 1.5;
I = f((a+b)/2)*(b-a)

patch([a a b b a],[0 f(b) f(b) 0 0],'y');
hold on
plot(x,y,[0.5 1],[f(b) f(b)],'black--');
legend('A','f(x)=x^3')
grid on
```

$I = 0.976562500000000$



5.2.3 Formula del trapezio

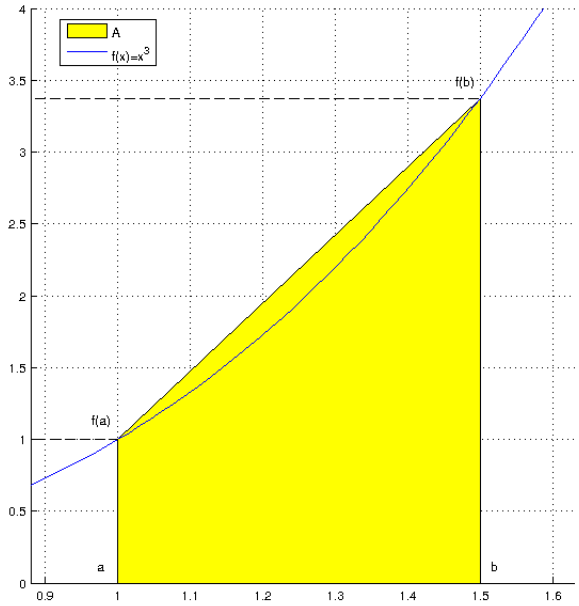
DEF 5.4 (formula del trapezio)**.** Per $n = 2$, $x_1 = a$ e $x_2 = b$:

$$\int_a^b f(x)dx \approx \int_a^b p_1(f;x)dx = \int_a^b \left[\frac{x-b}{a-b}f(a) + \frac{x-a}{b-a}f(b)\right]dx = \frac{b-a}{2}\big[f(a)+f(b)\big] \tag{5.5}$$

MATLAB 31. Codice Matlab

```
f = inline('x.^3');
x = linspace(0.5,2);
y = f(x);

a = 1;
b = 1.5;
I = (b-a)/2*(f(a)+f(b))

patch([a a b b a],[0 f(a) f(b) 0 0],'y');
hold on
plot(x,y, ...
     [0.5 1], [f(a) f(a)],'black--', ...
     [0.5 1.5], [f(b) f(b)],'black--');
legend('A','f(x)=x^3')
grid on
```

$I = \mathbf{1.0}93750000000000$



5.2.4 Formula di Simpson

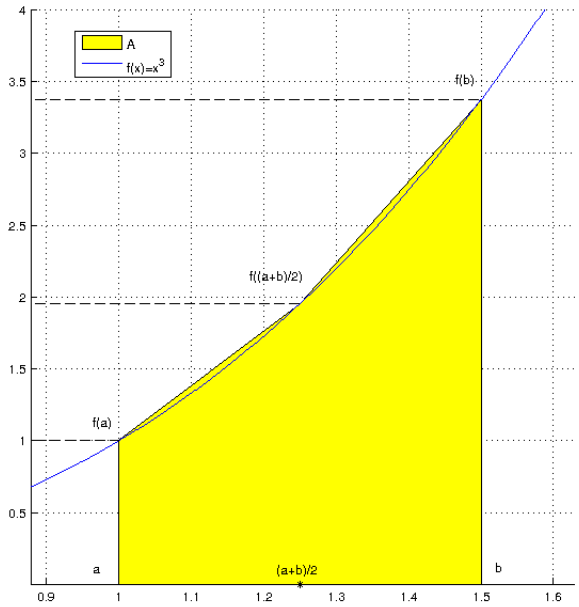
DEF 5.5 (formula di Simpson)**.** Per $n = 3$, $x_1 = a$, $x_2 = (a+b)/2$ e $x_3 = b$:

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b p_1(f;x)dx = \\ &= \int_a^b \left[\frac{(x-\frac{a+b}{2})(x-b)}{(a-\frac{a+b}{2})(a-b)}f(a) + \frac{(x-a)(x-b)}{(\frac{a+b}{2}-a)(\frac{a+b}{2}-b)}f\left(\frac{a+b}{2}\right) + \frac{(x-a)(x-\frac{a+b}{2})}{(b-a)(b-\frac{a+b}{2})}f(b)\right]dx = \\ &= \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]\end{aligned} \tag{5.6}$$

MATLAB 32. Codice Matlab

```
f = inline('x.^3');
x = linspace(0.5,2,1000);
y = f(x);

a = 1;
b = 1.5;
I = (b-a)/6*(f(a)+4*f((a+b)/2)+f(b))

patch([a a (a+b)/2 b b a],[0 f(a) f((a+b)/2) f(b) 0 0],'y');
hold on
plot(x,y, ...
     [0.5 1], [f(a) f(a)],'black--', ...
     [0.5 (a+b)/2], [f((a+b)/2) f((a+b)/2)],'black--', ...
     [0.5 1.5], [f(b) f(b)],'black--', ...
     (a+b)/2,0,'black*');
legend('A','f(x)=x^3')
grid on
```

$I =$ **1.015625**



5.3 Formule di interpolatura gaussiane

5.3.1 Definizioni

DEF 5.6 (formule di quadratura interpolatorie pesate)**.** Data una funzione $f(x)$ integrabile (ma non abbastanza regolare), se f è fattorizzabile nella forma:

$$f(x) = w(x)g(x) \tag{5.7}$$

dove $w(x)$ è una funzione di forma semplice chiamata funzione peso contenente le singolarità di f e $g(x)$ una funzione contenente la parte regolare di f, allora è possibile approssimare f approssimando con un polinomio solo la parte regolare di f:

$$\begin{aligned}\int_a^b w(x)g(x) &\approx \int_a^b w(x)p_{n-1}(g;x)dx = \\ &= \int_a^b w(x)\sum_{i=1}^n l_i(x)g(x_i)dx = \sum_{i=1}^n \left(\int_a^b w(x)l_i(x)dx\right)g(x_i) = \\ &= \sum_{i=1}^n \bar{w}_i g(x_i)\end{aligned} \tag{5.8}$$

DEF 5.7 (sistema di polinomi ortogonale alla funzione peso $w(x)$)**.** Data una funzione peso $w(x)$ non negativa nell'intervallo $[a,b]$ non identicamente nulla e tale che:

$$\int_a^b w(x)x^k dx < \infty \quad \forall k \tag{5.9}$$

un sistema di polinomi P_n si dice ortogonale in $[a,b]$ rispetto a $w(x)$ se:

$$\int_a^b w(x)P_n(x)P_m(x)dx \begin{cases} = 0 \text{ se } n \neq m \\ \neq 0 \text{ se } n = m \end{cases} \tag{5.10}$$

OSS 51. Per ogni n P_n possiede n zeri tutti reali, distinti e appartenenti ad $[a,b]$

DEF 5.8 (formula di interpolatura gaussiana)**.** Formula di quadratura interpolatoria pesata associata ai nodi x_i coincidenti con gli zeri dei polinomi ortogonali in $[a,b]$ rispetto alla funzione peso $w(x)$

5.3.2 Formula di Gauss-Legendre

DEF 5.9 (formula di Gauss-Legendre)**.** Per $w(x) = 1$, $[a,b] = [-1,1]$ e x_i^{GL} coincidenti con gli zeri del polinomio di grado n ortogonale in $[-1,1]$ rispetto a $w(x)$ (detti anche zeri di Legendre) si ha:

$$\int_{-1}^1 g(x)dx \approx \sum_{i=1}^n \bar{w}_i^{GL} g(x_i^{GL}) = \sum_{i=1}^n \left(\int_{-1}^1 l_i(x)dx\right)g(x_i^{GL}) \tag{5.11}$$

MATLAB 33. Codice Matlab
GaussLegendre.m

```
function f = GaussLegendre(g,a,b,xGL,wGL)
% xGL contiene i nodi della formula di GL
% wGL contiene i pesi della formula di GL
x = (b-a)/2*xGL+(b+a)/2;
y = g(x);
f = (b-a)/2*sum(wGL.*y);
end
```

5.3.3 Formula di Gauss-Chebyshev

DEF 5.10 (formula di Gauss-Chebyshev)**.** Per $w(x) = 1/\sqrt{1-x^2}$, $[a,b] = [-1,1]$ e x_i^{GC} coincidenti con gli zeri del polinomio di grado n ortogonale in $[-1,1]$ rispetto a $w(x)$ (detti anche zeri di Chebyshev) si ha:

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} g(x)dx \approx \sum_{i=1}^{n} \bar{w}_i^{GC} g(x_i^{GC}) = \frac{\pi}{n} \sum_{i=1}^{n} g\left(-\cos\left(\frac{2i-1}{2n}\pi\right)\right) \tag{5.12}$$

MATLAB 34. Codice Matlab
GaussChebyshev.m

```
function f = GaussChebyshev(g,n)
x = -cos((2*[1:n]-1)/(2*n)*pi);
y = g(x);
f = pi/n*sum(y);
end
```

5.3.4 Formula di Gauss-Jacobi

DEF 5.11 (formula di Gauss-Jacobi)**.** Per $w(x) = (1-x)^\alpha(1+x)^\beta$ con $\alpha, \beta > -1$, $[a,b] = [-1,1]$ e x_i^{GJ} coincidenti con gli zeri del polinomio di grado n ortogonale in $[-1,1]$ rispetto a $w(x)$ (detti anche zeri di Jacobi) si ha:

$$\int_{-1}^{1} (1-x)^\alpha(1+x)^\beta g(x)dx \approx \sum_{i=1}^{n} \bar{w}_i^{GJ} g(x_i^{GJ}) = \sum_{i=1}^{n} \left(\int_{-1}^{1} (1-x)^\alpha(1+x)^\beta l_i(x)dx \right) g(x_i^{GJ}) \tag{5.13}$$

OSS 52. È possibile utilizzare la formula di Gauss-Jacobi anche per il calcolo di integrali definiti nel generico intervallo $[a,b]$ mediante il cambiamento di variabile:

$$x = \frac{b-a}{2}t + \frac{b+a}{2} \tag{5.14}$$

da cui si ottiene:

$$\int_a^b (b-x)^\alpha(x-a)^\beta g(x)dx = \left(\frac{b-a}{2}\right)^{\alpha+\beta+1} \int_{-1}^{1} (1-t)^\alpha(1+t)^\beta g\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)dt \tag{5.15}$$

5.4 Errori e convergenza

5.4.1 Errori

5.4.1.1 Errore di interpolazione

DEF 5.12 (errore di interpolazione)**.** L'errore di interpolazione di una formula interpolatoria pesata è definito come:

$$E_n(g;x) = g(x) - p_{n-1}(g;x) \tag{5.16}$$

5.4.1.2 Errore di quadratura

DEF 5.13 (errore di quadratura)**.** L'errore di quadratura rispetto alla formula:

$$\int_a^b w(x)g(x) \approx \sum_{i=1}^{n} \bar{w}_i g(x_i) \tag{5.17}$$

è definito come:

$$R_n(g) = \int_a^b w(x)g(x)dx - \sum_{i=1}^{n} \bar{w}_i g(x_i) = \int_a^b w(x)E_n(g;x)dx \tag{5.18}$$

OSS 53. Fissato n le formule interpolatorie pesate sono esatte (cioè $R_n(g) \equiv 0$) se $g(x)$ è un polinomio di grado $n-1$

OSS 54. Le formule interpolatorie di Newton-Cotes a n nodi in un intervallo chiuso sono esatte se:

- n è pari e $f \in p_{n-1}$
- n è dispari e $f \in p_n$

5.4.2 Convergenza di una formula di quadratura

DEF 5.14 (convergenza di una formula di quadratura)**.** Una formula di quadratura si dice convergente se:

$$\lim_{n\to\infty} R_n(g) = 0 \tag{5.19}$$

TEOR 5.4.1. *Se:*

- $g \in C[a,b]$
- $\sum_{i=1}^{n} |\bar{w}_i| \leq M$ *con* M *indipendente da* n

allora:

$$\lim_{n\to\infty} R_n(g) = 0 \tag{5.20}$$

Inoltre se $g \in C^k([a,b])$ *allora per* $n \to \infty$*:*

$$|R_n(g)| = O\left(\frac{1}{n^k}\right) \tag{5.21}$$

OSS 55. Il teorema garantisce la convergenza delle formule gaussiane per ogni $g \in C([a,b])$

OSS 56. La convergenza delle formule di Newton-Cotes non è garantita nemmeno per $f \in C^\infty$

5.5 Formule di quadratura composte

5.5.1 Definizioni

PROP 5.5.1. *Per costruire una formula di quadratura composta per la risoluzione dell'integrale definito* $\int_a^b f(x)dx$ *è necessario:*

- *scegliere una formula di quadratura di base di cui si vuole costruire la versione composta*
- *suddividere l'intervallo di integrazione in* m *sottointervalli identificati da* x_i *(con* $a \equiv x_1$ *e* $b \equiv x_{m+1}$*)*
- *utilizzare la proprietà additiva degli integrali:*

$$\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_i}^{x_{i+1}} f(x)dx \tag{5.22}$$

- *approssimare l'integrale su ciscun sottointervallo mediante la formula scelta all'inizio*

5.5.2 Formula dei trapezi

DEF 5.15 (formula dei trapezi)**.** Utilizzando una partizione uniforme su un intervallo $[a, b]$ avente passo $h = (b-a)/m$ si individuano i punti $x_i = a + (i-1)h$ da cui:

$$\begin{aligned}\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_i}^{x_{i+1}} f(x)dx \approx \sum_{i=1}^{m} \frac{x_{i+1}-x_i}{2}[f(x_i)+f(x_{i+1})] = \\ = \frac{h}{2}[f(x_1)+f(x_2)+f(x_2)+f(x_3)+\cdots+f(x_m)+f(x_{m+1})] = \\ = \frac{h}{2}\left[f(x_1)+2\sum_{i=2}^{m} f(x_i)+f(x_{m+1})\right]\end{aligned} \tag{5.23}$$

MATLAB 35. Codice Matlab
trapezi.m

```
function g = trapezi(f,a,b,m)
x = linspace(a,b,m+1);
y = f(x);
g = (b-a)/(2*m)*(y(1)+2*sum(y(2:m))+y(m+1));
end
```

ex.m

```
a = 0;
b = 1;
n = 15;
m = 2^n
f = inline('x.^3.*exp(x)');
trapezi(f,a,b,m)
```

$m = 32768$
$ans = \mathbf{0.563436343}925772$

OSS 57. Se f è periodica e il periodo è un sottomultiplo di $b-a$ allora l'ordine di convergenza è:

$$|R_m(f)| = O(h^{2k+1}) \tag{5.24}$$

5.5.3 Formula di Simpson

DEF 5.16 (formula di Simpson)**.** Utilizzando una partizione uniforme su un intervallo $[a, b]$ avente passo $h = (b-a)/(2m)$ si individuano i punti $x_i = a + (i-1)h$ da cui:

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{i=1}^{m} \int_{x_{2i-1}}^{x_{2i+1}} f(x)dx \approx \sum_{i=1}^{m} \frac{x_{2i+1} - x_{2i-1}}{6}[f(x_{2i-1}) + 4f(x_{2i}) + f(x_{2i+1})] = \\ &= \frac{h}{3}[f(x_1) + 4f(x_2) + f(x_3) + \cdots + f(x_{2m-1}) + 4f(x_{2m}) + f(x_{2m+1})] = \\ &= \frac{h}{3}\left[f(x_1) + 4\sum_{i=1}^{m} f(x_{2i}) + 2\sum_{i=1}^{m-1} f(x_{2i+1}) + f(x_{2m+1})\right]\end{aligned} \tag{5.25}$$

MATLAB 36. Codice Matlab
simpson.m

```
function g = simpson(f,a,b,m)
x = linspace(a,b,2*m+1);
y = f(x);
g = (b-a)/(6*m)*(y(1)+4*sum(y(2:2:2*m))+2*sum(y(3:2:2*m-1))+y(2*m+1))
    ;
end
```

ex.m

```
a = 0;
b = 1;
n = 8;
m = 2^n
f = inline('x.^3.*exp(x)');
simpson(f,a,b,m)
```

$m = 256$
$ans = \mathbf{0.56343634308}8896$

OSS 58. Se $f \in C^4([a, b])$ allora l'ordine di convergenza è:

$$|R_m(f)| = O(h^4) \tag{5.26}$$

5.5.4 Convergenza

PROP 5.5.2. *Le formule composte convergono per* $m \to \infty$ *ovvero per* $h \to 0 \quad \forall f \in C([a, b])$

5.5.5 Comandi Matlab

5.5.5.1 *quad* e *quadl*

DEF 5.17 (formula di quadratura composta con partizione adattiva)**.** Formula di quadratura composta in cui la partizione non è uniforme ma ha un addensamento di nodi in prossimità delle singolarità della funzione

MATLAB 37. Codice Matlab

```
a = 0;
b = 1;
toll = 1.0*10^(-14);
f = inline('x.^3.*exp(x)');

[q,N] = quad(f,a,b,toll)
[q,N] = quadl(f,a,b,toll)
```

$q = \mathbf{0.563436343081909}$
$N = 1025$
$q = \mathbf{0.563436343081910}$
$N = 48$

Indice analitico

List of Algorithms