Corsi di Laurea in
Ingegneria Biomedica e Ingegneria Elettronica

Note di Fondamenti di Informatica I (II semestre)

Silvio Sabatini

Indice

Parte I

Fondamenti di calcolo numerico e applicazione in semplici problemi di Ingegneria

Capitolo 1

Introduzione al calcolo numerico

La trattazione dei semplici metodi ed algoritmi di calcolo numerico presentata nel seguito in queste note non richiedono al lettore specifiche competenze teoriche sull'argomento. Tuttavia, riteniamo utile in questo capitolo introdurre alcune problematiche legate alle elaborazioni numeriche sui calcolatori. In particolare, consideriamo (i) le approssimazioni dovute all'aritmetica a precisione finita dei calcolatori e alla discretizzazione dei problemi analitici, e (ii) l'analisi degli errori associati a tali approssimazioni.

1.1 Rappresentazione dei numeri in un calcolatore

Parlando di numeri ed operazioni in un calcolatore, sorge l'interrogativo su quante cifre si possono utilizzare per la loro rappresentazione. Noi siamo abituati a non porci tale problema (basta aggiungere cifre!!), ma in un calcolatore il numero di dispostivi hardware che possono contenere e manipolare le cifre è finito. Pertanto, siamo costretti a riservare alla rappresentazione di ogni numero uno "spazio" finito di memoria costituito da una sequenza finita di *bit* (*binary digit*) o *byte* (gruppi di 8 bit).

Ogni calcolatore permette al programmatore di scegliere tra diverse convenzioni di rappresentazione o *tipi di dato*. I tipi di dato possono differire nel numero di bit utilizzati (lunghezza della parola), ma anche nel formato di rappresentazione. Si noti comunque, che qualunque sia la codifica prescelta, la rappresentazione dei numeri reali nel calcolatore è soggetta ad approssimazioni; tali approssimazioni, come vedremo, si propagano nel corso dell'esecuzione delle operazioni, causando errori numerici anche importanti. Il **calcolo numerico** è la disciplina che studia le proprietà dell'esecuzione delle operazioni tramite calcolatore, consentendo di valutare l'entità degli errori numerici introdotti durante l'esecuzione delle operazioni.

Considerato il numero finito di byte disponibili per la codifica dei numeri in un calcolatore, possiamo rappresentare solo quei numeri che, in base alla convenzione di rappresentazione scelta, possono essere contenuti nello spazio previsto per ciascuno di essi; questi numeri sono detti *numeri di macchina*.

1.1.1 Numeri interi

La rappresentazione di un numero intero comporta la rappresentazione del segno del numero oltre al suo valore. Se n bit sono disponibili per rappresentare un intero, la rappresentazione in modulo e segno[1] utilizza il primo bit (la posizione più a sinistra) come bit di segno; per convenzione, 0 indica un numero positivo e 1 un numero negativo. Con questa convenzione, quando sono disponibili n bit come lunghezza fissata per contenere i dati, possono essere rappresentati gli interi compresi fra $-(2^{n-1}-1)$ e $+(2^{n-1}-1)$. Infatti, essendo un bit utilizzato per la rappresentazione del segno, solo $n-1$ bit sono disponibili per rappresentare il modulo del numero (v. Tab. 1.1).

Tipo	**Allocazione in memoria**	**Accuratezza**	**Insieme di rappresentazione**
`signed int`	2	-	$\{-2^{15}-1, 2^{15}-1\}$
`unsigned int`	2	-	$\{0, 2^{16}-1\}$
`float`	4	6 cifre dec.	$\pm\{10^{-38}, 10^{38}\} \cup \{0\}$
`double`	8	15 cifre dec.	$\pm\{10^{-308}, 10^{308}\} \cup \{0\}$

Tabella 1.1: I valori riportati sono solo indicativi e possono variare da macchina a macchina

Compatibilmente con in numero di bit (cifre) disponibili, la rappresentazione di un numero intero di un calcolatore risulta esatta. Anche l'aritmetica sui numeri interi risulta esatta osservando tuttavia che (i) il risultato di una divisione tra interi produce ancora un intero, con conseguente perdita della parte frazionaria del risultato (del resto della divisione) e che (ii) il risultato può essere al di fuori dell'intervallo di rappresentazione. Ad esempio, con numeri a 8 bit, si potrebbe verificare un inconveniente come quello qui evidenziato:

```
  0000 0001  +          1  +
  1111 1111  =        255  =
1 0000 0000             0
```

in cui la prima cifra, che è la più significativa, viene inesorabilmente persa portando a un risultato manifestamente errato. Tale situazione prende il nome di *overflow*.

Per comprendere le proprietà dell'aritmetica finita è utile far ricorso ad una rappresentazione grafica degli interi analoga a quella, basata su una retta, utilizzata per l'aritmetica comune. Sulla retta di Fig. 1.1 sono rappresentati gli interi: un'operazione di somma tra numeri positivi consiste nello spostarsi su tale retta, verso destra a partire dal primo addendo, di un segmento di lunghezza pari al valore del secondo addendo. Lavorando in aritmetica finita, se si supera il limite della rappresentazione si ritorna al valore iniziale, come nell'esempio precedente. Quindi, la rappresentazione non sarà più una retta, bensì

[1]Esistono diverse altre codifiche binarie di numeri interi. Tra queste, una delle più utilizzate per la semplificazione dell'esecuzione di operazioni aritmetiche tra numeri interi, è la *rappresentazione in complemento a 2*.

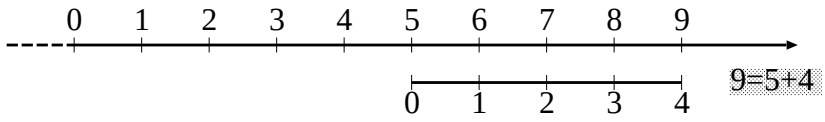


Figura 1.1: La retta dei numeri interi

una circonferenza. La Fig. 1.2 illustra la rappresentazione binaria su 3 bit. In essa, uno spostamento di un segmento di lunghezza 4 a partire da 5 (`101`) porterebbe ad attraversare lo zero, arrivando a 1. Infatti il risultato corretto, 9, non è rappresentabile su 3 bit e l'operazione dà luogo a overflow.

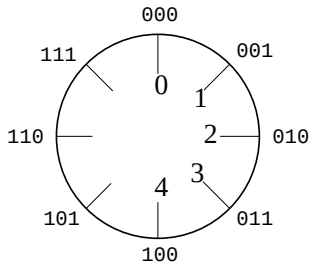


Figura 1.2: Rappresentazione degli interi in aritmetica finita

1.1.2 Numeri reali

In generale, un numero reale può essere rappresentato solo in forma approssimata con arbitraria precisione mediante allineamenti finiti di cifre. Dovendo utilizzare un calcolatore digitale per calcoli scientifici di massima precisione, con valori molto grandi o molto piccoli, il principale problema da affrontare è la limitattezza dello spazio (cioè il numero di bit) a disposizione per rappresentare i numeri. Naturalmente, la strada da seguire non è quella di aumentare arbitrariamente il numero di bit impiegati per rappresentare ciascun numero. Si ricorre invece a un particolare tipo di notazione denominata in *virgola mobile* (*floating point*) che utilizza la notazione esponenziale per la codifica dei numeri reali; per questa rappresentazione non esiste un unico standard, perciò consideriamo una delle tante possibili convenzioni. Secondo tale rappresentazione, ogni numero reale x può essere scritto nella forma:

$$x = mB^e$$

dove m è un numero reale, detto *mantissa*, B è la *base* del sistema di numerazione scelto ($B = 10$ per il sistema decimale, $B = 2$ per il sistema binario), ed e è un *esponente* intero positivo, negativo o nullo. Questa rappresentazione non è unica; infatti abbiamo

$$x = mB^e = m'B^{e-1} = m''B^{e+1} = \ldots, \qquad m' = mB, \;\; m'' = m/B.$$

Si noti che l'insieme dei numeri generabili con questa notazione è un sottoinsieme dei numeri reali, distribuito in modo non uniforme; precisamente, vi saranno valori estremamente vicini

tra loro nell'intorno del numero 0 ed estremamente lontani fra loro nell'intorno del numero massimo esprimibile, positivo o negativo.

La notazione esponenziale consente di scrivere in poco spazio numeri molto grandi o molto piccoli evitando di sprecare un numero elevato di bit con valori nulli atti soltanto ad indicare l'ordine di grandezza del numero. Aumentando così il numero di bit significativi, aumenta la *precisione* del numero. In pratica, nei calcolatori digitali i byte destinati a memorizzare un numero reale sono così suddivisi:

- un bit per il segno della mantissa (e quindi dell'intero numero);
- t bit per la mantissa ($m_i < m < m_s$);
- q bit per l'esponente ($M_i < e < M_s$).

La rappresentazione di $x \neq 0$ si dice normalizzata quando

$$B^{-1} \leq |m| < 1 \, .$$

Fissata la base B, ogni numero reale $x \neq 0$ è univocamente definito dalla coppia (m, e), per cui è sufficiente memorizzare quest'ultima. Non esistendo la rappresentazione normalizzata del numero $x = 0$, per convenzione si individua tale numero con la coppia $(0, 0)$ (v. Tab. 1.1).

Le operazioni su numeri reali rappresentati con un numero finito di cifre possono dare luogo a fenomeni di *overflow* (numeri con esponente $e > M_s$) o di *underflow* (numeri con esponente $e < M_i$).

1.2 Approssimazioni ed errori

1.2.1 Problemi numerici e algoritmi

Per **problema numerico** intendiamo una descrizione chiara e non ambigua di una relazione funzionale tra i dati (*input*) del problema (che costituiscono le variabili indipendenti) e i risultati desiderati (*output*) (v. Fig. 1.3). I dati x e i risultati y devono essere rappresentabili da vettori (di dimensione finita) di numeri. La relazione funzionale f può essere espressa sia in forma esplicita $y = f(x)$, sia implicita $f(x, y) = 0$.

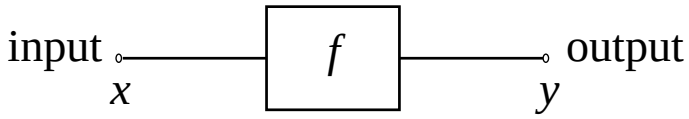


Figura 1.3: Schematica rappresentazione di un problema numerico

Per **algoritmo** di un problema numerico intendiamo una descrizione completa e ben definita di operazioni che permetta di trasformare (in un numero finito di passi) ogni

vettore di dati (permissibili) x nel corrispondente output y^*, non necessariamente uguale a y. Ad ogni problema numerico possiamo associare più algoritmi, ognuno dei quali in genere fornirà risultati con precisione diversa.

Il problema numerico è caratterizzato dalla sola relazione funzionale esistente tra input e output; in un algoritmo invece, ogni singolo passo (operazione) deve essere definito chiaramente ed inequivocabilmente, dai dati sino alla determinazione effettiva dei risultati.

Nei capitoli che seguono parleremo di problemi e di algoritmi; in particolare esamineremo algoritmi per il calcolo della soluzione di problemi numerici spesso ottenuti approssimando (o meglio, *discretizzando*) modelli matematici.

Generalmente, tra la soluzione analitica e quella numerica, si osserva una discrepanza, o *errore*, dovuta al fatto che le tecniche numeriche presuppongono un'approssimazione. Errori di questo tipo sono caratteristici della maggior parte delle tecniche descritte in queste note. Ciò può sembrare a prima vista contrario rispetto a quanto ci si aspetta da una tecnica ingegneristica valida. Gli studenti e i consulenti di ingegneria, per esempio, tentano costantemente di limitare gli errori nel proprio lavoro: quando lo studente sostiene esami o esegue esercizi, viene penalizzato per gli errori commessi. Nella pratica professionale, poi, gli errori possono costare parecchio e possono risultare addirittura catastrofici: il cedimento di una struttura o di una macchina, ad esempio, può comportare la perdita di vite umane.

Sebbene la perfezione sia un traguardo a cui mirare, essa, in pratica, non viene mai raggiunta. I metodi numerici possono introdurre simili discrepanze nell'analisi; anche in questo caso la domanda è quale sia l'errore tollerabile. Nel seguito sono trattati gli argomenti fondamentali relativi all'identificazione, quantificazione e minimizzazione di tali errori. In particolare verranno descritti i due tipi principali di errori: di arrotondamento e di troncamento. Gli *errori di arrotondamento* sono dovuti al fatto che i calcolatori dispongono di un numero finito di cifre per rappresentare i dati. L'*errore di troncamento* è la discrepanza introdotta dal fatto che i metodi numerici utilizzano delle approssimazioni per eseguire calcoli matematici e rappresentare quantità esatte. Per esempio, moltissimi metodi numerici richiederebbero l'esecuzione di una sequenza infinita di passi di calcolo: essi sono perciò "troncati" dopo un numero di passi finito, quando si è raggiunta un'approssimazione sufficiente dei risultati.

1.2.2 Definizione degli errori

Gli *errori numerici* sono dovuti all'uso di approssimazioni nella rappresentazione di quantità esatte e nell'esecuzione di calcoli matematici. Gli *errori di troncamento* dipendono dal fatto che una certa procedura matematica viene approssimata da una serie di operazioni aritmetiche più semplici, mentre gli *errori di arrotondamento* risultano dall'uso di un numero limitato di cifre significative per la rappresentazione dei numeri. In entrambi i casi, la relazione tra il risultato esatto, o vero, e l'approssimazione può essere così formulata:

$$\text{valore vero} = \text{approssimazione} + \text{errore}$$

da cui troviamo che l'errore numerico è uguale alla differenza tra il valore vero e l'approssimazione:

$$e = \text{valore vero} - \text{approssimazione} \tag{1.1}$$

dove e indica l'*errore assoluto*.

Questa definizione presenta uno svantaggio: non tiene in nessun conto l'ordine di grandezza del valore che si sta esaminando. Per esempio, un errore di un centimetro è molto più significativo quando si misura la lunghezza di un chiodo piuttosto che quella di un ponte. Per tener conto dell'ordine di grandezza delle quantità in esame si può normalizzare l'errore rispetto al valore vero:

$$\epsilon = \frac{\text{errore}}{\text{valore vero}} = \frac{e}{\text{valore vero}}$$

L'errore relativo può anche essere moltiplicato per 100 in modo da esprimerlo come frazione di 100 e non di 1:

$$\epsilon_p = \frac{\text{errore}}{\text{valore vero}} 100\%$$

dove ϵ_p viene definito *errore relativo percentuale*.

Nel caso dei metodi numerici, il valore vero è noto solo quando si ha a che fare con funzioni che possono essere espresse analiticamente, come nello studio teorico del comportamento di una particolare tecnica. Nelle applicazioni reali, invece, il risultato esatto non è mai noto a priori: in questi casi, l'alternativa è normalizzare l'errore assoluto usando la migliore stima disponibile del valore vero; normalizzare, cioè, rispetto all'approssimazione stessa

$$\epsilon_a = \frac{\text{valore vero} - \text{approssimazione}}{\text{approssimazione}} \tag{1.2}$$

dove il pedice a significa che l'errore è normalizzato rispetto a un valore approssimato. Nelle applicazioni reali, inoltre, la (1.1) non può essere usata per calcolare il termine relativo all'errore contenuto nella (1.2): una delle difficoltà che si incontrano nell'applicazione dei metodi numerici sta nella determinazione di stime dell'errore in mancanza di nozioni circa il valore vero. Certi metodi numerici, per esempio, arrivano a un risultato procedendo per iterazioni; secondo questo tipo di approccio, l'approssimazione più recente viene calcolata sulla base della precedente. Questo processo viene ripetuto, o iterato, per calcolare approssimazioni sempre migliori. In questi casi, la stima dell'errore che viene assunta è semplicemente la differenza tra il risultato dell'ultima iterazione e quello della precedente. L'errore relativo, quindi, viene determinato secondo la formula

$$\epsilon_a = \frac{\text{approssimazione attuale} - \text{approssimazione precedente}}{\text{approssimazione attuale}} \tag{1.3}$$

Questo e altri approcci per la stima degli errori verranno utilizzati nei prossimi capitoli. I valori che si ottengono dalle espressioni (1.1)-(1.3) possono essere sia positivi che negativi; se l'approssimazione è maggiore del valore vero (o la penultima approssimazione è maggiore dell'ultima), l'errore è negativo; in caso contrario, l'errore è positivo. Inoltre per le espressioni dell'errore relativo, il denominatore può essere minore di zero e anche questa

condizione può portare a un errore negativo. Spesso, quando si eseguono dei calcoli il segno dell'errore può non essere interessante, ma ciò che realmente importa è che il valore assoluto dell'errore sia minore di una tolleranza ϵ_s specificata in partenza. Pertanto, è spesso utile fare riferimento al valore assoluto delle espressioni (1.1)-(1.3); in tali casi, il calcolo viene proseguito finché si ha:

$$|\epsilon_a| < \epsilon_s$$

Se questa condizione è verificata, si assume che il risultato sia corretto entro la tolleranza ϵ_s prefissata.

1.2.3 Errori di arrotondamento, operazioni di macchina

Se x rappresenta un valore esatto e $\bar{x}$ una sua approssimazione, l'errore assoluto e l'errore relativo associati a $\bar{x}$ sono definiti rispettivamente dalle quantità:

$$|x - \bar{x}| \quad \text{e} \quad \left|\frac{x - \bar{x}}{x}\right|, \quad x \neq 0;$$

inoltre, possiamo scrivere

$$\bar{x} = x(1 + \epsilon), \qquad \epsilon = \frac{\bar{x} - x}{x}.$$

Quando

$$|x - \bar{x}| \leq \frac{1}{2} B^{-k}, \qquad k \geq 1,$$

diciamo che l'approssimazione $\bar{x}$ ha k "decimali" corretti (nella base B), e definiamo *significative* le cifre che precedono il $(k+1)$-esimo decimale (escludendo gli eventuali zeri iniziali). Le cifre significative coincidono con i decimali corretti presenti nella mantissa $\bar{m}$ della rappresentazione (normalizzata) $\bar{x} = \bar{m} B^e$. Se la mantissa $\bar{m}$ ha k decimali corretti possiamo scrivere:

$$|m - \bar{m}| \leq \frac{1}{2} B^{-k};$$

quindi, ricordando che $|m| < 1$, se vale una disuguaglianza del tipo

$$\frac{|x - \bar{x}|}{|x|} \leq \frac{1}{2} B^{-k}$$

possiamo senz'altro affermare che l'approssimazione $\bar{x}$ ha almeno k cifre significative. Infatti, dall'ultima relazione deduciamo

$$|m - \bar{m}| \leq \frac{1}{2} B^{-k} |m| < \frac{1}{2} B^{-k}.$$

Diciamo "almeno k cifre" perché in realtà secondo la nostra definizione potrebbero essere $k + 1$.

Come abbiamo visto nel paragrafo 1.1.2, in un elaboratore non tutti i numeri reali possono venire rappresentati. Nel caso di un sistema floating-point con t cifre (nella base B)

per la mantissa, tutti i numeri che nella base scelta ammettono una rappresentazione (normalizzata) con un numero di cifre nella mantissa superiore a t dovranno in qualche modo venire "accorciati", o meglio, arrotondati a t cifre. Le operazioni di arrotondamento usate sono essenzialmente due:

(i) $T_t(x) = B^{-t}\lfloor xB^t \rfloor$ *"trunc"*,
esclude la parte a destra della t-esima cifra;

(ii) $R_t(x) = B^{-t}\lfloor xB^t + \frac{1}{2} \rfloor$ *"round"*,
aggiunge $\frac{1}{2}B^{-t}$ alla mantissa in questione e poi tronca quest'ultima alla t-esima cifra;

dove $\lfloor x \rfloor$ è l'operatore "parte intera" del numero reale x:

$$\lfloor x \rfloor = \max\{n \in Z \quad : \quad n \leq x\}$$

che restituisce il più grande intero più piccolo di x.

Esempio. Supponendo $t = 6$ e $B = 10$, la mantissa

$$m = 0.74749563$$

verrebbe arrotondata a

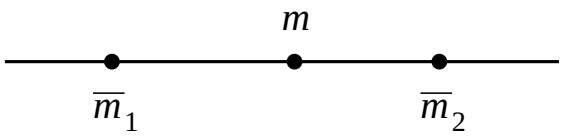
$$\bar{m} = 0.747495$$

adottando la tecnica (i), e a

$$\bar{m} = 0.747496$$

con la (ii).

Le mantisse $\bar{m}$ dei numeri di macchina, $B^{-1} \leq |\bar{m}| < 1$, non hanno più di t cifre, e la distanza tra due mantisse di macchina consecutive è esattamente B^{-t}. Con l'operatore *trunc*, tutte le mantisse m comprese tra due mantisse di macchina consecutive $\bar{m}_1$ e $\bar{m}_2 = \bar{m}_1 + B^{-t}$, positive per esempio,



vengono sostituite da $\bar{m}_1$; in questo caso abbiamo $|m - \bar{m}_1| < B^{-t}$. Con l'operatore *round* invece, tutte le mantisse comprese nell'intervallo $(\bar{m}_1, \bar{m}_1 + \frac{1}{2}B^{-t})$ vengono sostituite con $\bar{m}_1$, mentre le mantisse situate in $[\bar{m}_1 + \frac{1}{2}B^{-t}, \bar{m}_2)$ vengono approssimate con $\bar{m}_2$, così che, denotando con $\bar{m}$ la mantissa di macchina che il criterio di arrotondamento associa a m, risulta $|m - \bar{m}| \leq \frac{1}{2}B^{-t}$.

Dei due metodi, il primo è certamente il peggiore (ma il meno oneroso), non solo perché può provocare un errore maggiore (doppio rispetto al secondo), ma anche perché il segno di $(m - \bar{m})/m$ è costante (positivo).

Sia $x = mB^e$, $B^{-1} \leq |m| < 1$, un numero reale, e sia $\bar{x} = \bar{m}B^e$ il corrispondente numero di macchina ottenuto mediante una delle due tecniche di arrotondamento introdotte. Poiché

$$|m - \bar{m}| \leq \begin{cases} B^{-t} & \text{con la (i)} \\ \frac{1}{2}B^{-t} & \text{con la (ii)}, \end{cases}$$

per gli errori, assoluto e relativo, associati ad $\bar{x}$ abbiamo rispettivamente

$$|x - \bar{x}| \leq \begin{cases} B^{e-t} & \text{(i)} \\ \frac{1}{2}B^{e-t} & \text{(ii)}, \end{cases}$$

e

$$\frac{|x - \bar{x}|}{|x|} \leq \frac{|x - \bar{x}|}{B^{e-1}} \leq \begin{cases} B^{1-t} & \text{(i)} \\ \frac{1}{2}B^{1-t} & \text{(ii)}. \end{cases}$$

La quantità $\varepsilon_m = B^{1-t}$ nel caso (i), e $\varepsilon_m = \frac{1}{2}B^{1-t}$ in (ii), definisce la cosidetta *precisione di macchina*. Essa è una costante caratteristica di ogni aritmetica floating-point (e tecnica di arrotondamento), e rappresenta la massima precisione di calcolo raggiungibile con il calcolatore su cui tale aritmetica è implementata. Non ha pertanto senso cercare di determinare approssimazioni con precisione relativa inferiore alla quantità ε_m. In altre parole la precisione di macchina è legata al massimo numero di cifre significative relative all'ordine di grandezza dell'esponente. Tipicamente, un calcolatore con $B = 2$ e una lunghezza di parola di 32 bit ha ε_m circa pari a $3 \cdot 10^{-8}$. Si noti infine, che ε_m non corrisponde al più piccolo numero rappresentabile nella macchina. Tale numero dipende infatti dal numero di bit che rappresentano l'esponente, mentre ε_m dipende dal numero di bit che rappresentano la mantissa.

Nonostante la tecnica di arrotondamento (ii) sia migliore, per motivi di semplicità, costi e velocità, in molti calcolatori viene implementata la tecnica (i).

Finora abbiamo definito i numeri di macchina e visto come arrotondare un generico numero reale a numero di macchina. Osserviamo tuttavia che i risultati di operazioni aritmetiche tra numeri di macchina generalmente non sono numeri di macchina; pertanto in un calcolatore risulterà impossibile implementare esattamente le operazioni aritmetiche. Dovremo accontentarci di realizzare le cosidette *"operazioni di macchina"*. L'operazione di macchina associa a due numeri di macchina un terzo numero di macchina, ottenuto arrotondando (con le tecniche (i) o (ii)) l'esatto risultato dell'operazione aritmetica in questione.

Se con $\bar{x} = \text{fl}(x)$ indichiamo l'operazione di arrotondamento, in aritmetica floating-point, di x a numero macchina $\bar{x}$ [2], e con $\oplus$, $\ominus$, $\otimes$, $\oslash$ denotiamo le operazioni di macchina

[2]Ricordiamo che $\bar{x} = x(1 + \epsilon)$, $|\epsilon| \leq \varepsilon_m$.

corrispondenti a quelle aritmetiche $+$, $-$, $\times$, $/$, abbiamo

$$\begin{aligned}
\bar{x} \oplus \bar{y} &= \mathrm{fl}(\bar{x} + \bar{y}) = (\bar{x} + \bar{y})(1 + \epsilon_1) \\
\bar{x} \ominus \bar{y} &= \mathrm{fl}(\bar{x} - \bar{y}) = (\bar{x} - \bar{y})(1 + \epsilon_2) \\
\bar{x} \otimes \bar{y} &= \mathrm{fl}(\bar{x} \times \bar{y}) = (\bar{x} \times \bar{y})(1 + \epsilon_3) \\
\bar{x} \oslash \bar{y} &= \mathrm{fl}(\bar{x} / \bar{y}) = (\bar{x} / \bar{y})(1 + \epsilon_4)
\end{aligned}$$

dove $|\epsilon_i| \leq \varepsilon_m$. L'errore relativo introdotto dalle quattro operazioni aritmetiche di macchina, prescindendo dagli eventuali errori presenti nei due operandi $\bar{x}$ e $\bar{y}$, non supera mai la precisione di macchina ε_m.

E' importante osservare che per le operazioni di macchina non valgono le note proprietà (commutativa, associativa, distributiva, ecc.) delle quattro operazioni aritmetiche. In particolare, mentre la proprietà commutativa

$$\bar{x} \oplus \bar{y} = \bar{y} \oplus \bar{x} \ , \qquad \bar{x} \otimes \bar{y} = \bar{y} \otimes \bar{x}$$

risulta ancora valida, le seguenti <u>non</u> lo sono più:

$$\begin{aligned}
&\bar{x} \oplus (\bar{y} \oplus \bar{z}) = (\bar{x} \oplus \bar{y}) \oplus \bar{z} \ , \qquad \bar{x} \otimes (\bar{y} \otimes \bar{z}) = (\bar{x} \otimes \bar{y}) \otimes \bar{z}, \\
&\bar{x} \otimes (\bar{y} \oplus \bar{z}) = (\bar{x} \otimes \bar{y}) \oplus (\bar{x} \otimes \bar{z}), \\
&(\bar{x} \otimes \bar{y}) \oslash \bar{y} = \bar{x}, \qquad (\bar{x} \oslash \bar{y}) \otimes \bar{y} = \bar{x}, \\
&(\bar{x} \otimes \bar{z}) \oslash \bar{y} = (\bar{x} \oslash \bar{y}) \otimes \bar{z}.
\end{aligned}$$

Un'ulteriore relazione anomala è la seguente:

$$\bar{x} \oplus \bar{y} = \bar{x} \qquad \text{quando } \ |\bar{y}| < \frac{\varepsilon_m}{B} |\bar{x}|.$$

Possiamo pertanto concludere che *le espressioni che sono equivalenti in aritmetica esatta non risultano generalmente tali nelle aritmetiche con precisione finita.* Ciononostante due espressioni (non nulle) saranno definite "equivalenti" dal punto di vista del calcolo numerico quando, valutate in un calcolatore, forniscono risultati che differiscono per una tolleranza relativa dell'ordine della precisione di macchina.

Osservazione. Anche se i dati iniziali di un processo di calcolo sono numeri di macchina, o comunque arrotondabili a numeri di macchina, le operazioni intermedie possono dare origine a fenomeni di *overflow* (numeri con esponente $e > M_s$) o di *underflow* (numeri con $e < M_i$); quando ciò accade, i risultati di tali operazioni non sono rappresentabili ed il calcolatore invia una segnalazione di errore.

Esempi.

$$\begin{aligned}
&z = \sqrt{x \otimes y}, \qquad x = m_1 B^{e_1}, \ \ y = m_2 B^{e_2}, \\
&\text{overflow se } \ e_1, e_2 > \frac{M_s}{2}, \\
&\text{underflow se } \ e_1, e_2 < \frac{M_i}{2};
\end{aligned}$$

$z = \sqrt{x \oslash y}$
overflow se $e_1 > 0, e_2 < 0, e_1 - e_2 > M_s$,
underflow se $e_1 < 0, e_2 > 0, e_1 - e_2 < M_i$.

Propagazione degli errori

Vediamo ora come un errore può propagarsi nell'esecuzione di calcoli in successione. Consideriamo l'addizione di due numeri x e y (i loro valori veri) usando i valori approssimati $\bar{x}$ e $\bar{y}$ a cui associamo rispettivamente gli errori assoluti e_x e e_y. Cominciando con $x = \bar{x} + e_x$ e $y = \bar{y} + e_y$, la somma è

$$x + y = (\bar{x} + e_x) + (\bar{y} + e_y) = (\bar{x} + \bar{y}) + (e_x + e_y) \ .$$

Pertanto, per l'addizione, l'errore nel risultato è la somma degli errori degli addendi.

La propagazione dell'errore nella moltiplicazione è più complicata. Il prodotto è

$$xy = (\bar{x} + e_x)(\bar{y} + e_y) = \bar{x}\bar{y} + \bar{x}e_y + \bar{y}e_x + e_x e_y \ . \tag{1.4}$$

Pertanto, se $\bar{x}$ e $\bar{y}$ sono maggiori di 1 in valore assoluto, i termini $\bar{x}e_y$ e $\bar{y}e_x$ indicano che esiste la possibilità di amplificare gli errori iniziali e_x e e_y. Il fenomeno risulta più evidente se esaminiamo l'errore relativo. Riscrivendo la relazione (1.4) si ottiene

$$xy - \bar{x}\bar{y} = \bar{x}e_y + \bar{y}e_x + e_x e_y \ . \tag{1.5}$$

Supponiamo che $x \neq 0$ e $y \neq 0$, allora possiamo dividere la (1.5) per xy e ottenere

$$\begin{aligned} \frac{xy - \bar{x}\bar{y}}{xy} &= \frac{\bar{x}e_y + \bar{y}e_x + e_x e_y}{xy} \\ &= \frac{\bar{x}e_y}{xy} + \frac{\bar{y}e_x}{xy} + \frac{e_x e_y}{xy} \ . \end{aligned}$$

Inoltre, supponiamo che $\bar{x}/x \approx 1$, $\bar{y}/y \approx 1$, e $(e_x/x)(e_y/y) = \epsilon_x \epsilon_y \approx 0$. Allora facendo queste sostituzioni otteniamo la seguente relazione semplificata:

$$\frac{xy - \bar{x}\bar{y}}{xy} \approx \frac{e_y}{y} + \frac{e_x}{x} + 0 = \epsilon_y + \epsilon_x \ .$$

Questa relazione mostra che l'errore relativo sul prodotto xy è approsimativamente la somma degli errori relativi sulle approssimazioni $\bar{y}$ e $\bar{x}$.

Cancellazione numerica

La conseguenza più grave della rappresentazione con precisione finita dei numeri reali è senza dubbio il fenomeno della cancellazione numerica, ovvero la *perdita di cifre significative* dovuta ad operazioni di sottrazione quando il risultato è più piccolo di ciascuno dei due operandi; questo fenomeno si verifica quando i due operandi sono "quasi uguali". Per

meglio illustrare quanto accade, supponiamo di avere due numeri floating-point $a = m_1 B^q$ e $b = m_2 B^q$, dove le mantisse m_1 e m_2, pur avendo più di t cifre, sono rappresentabili solo con t cifre. Supponiamo inoltre che le prime tre cifre, per esempio, di m_1 coincidano con le corrispondenti di m_2. Nella mantissa $\bar{m}$ della differenza (normalizzata) $\bar{a} \ominus \bar{b} = \bar{m} B^{q-3}$, solamente le prime $t-3$ cifre provengono dalle mantisse m_1 e m_2; le restanti 3 (poste uguali a zero) non hanno alcun significato. Consideriamo il seguente esempio numerico:

$$B = 10,\ t = 6 \text{ e tecnica di arrotondamento } (\textit{round})$$
$$m_1 = .147554326, \qquad \bar{m}_1 = .147554$$
$$m_2 = .147251742, \qquad \bar{m}_2 = .147252.$$

La mantissa $\bar{m}$ della differenza di macchina risulta

$$\bar{m} = (\bar{m}_1 \ominus \bar{m}_2) \times 10^3 = .302000\ ,$$

mentre la vera mantissa è

$$m = (m_1 - m_2) \times 10^3 = .302584\ .$$

L'operazione di sottrazione in sè, anche quella di macchina, non introduce alcuna perdita di precisione; come abbiamo visto nel paragrafo precedente, le quattro operazioni aritmetiche di macchina possono provocare un errore relativo che non supera mai la precisione di macchina. La perdita di cifre significative descritta nell'esempio precedente (dove peraltro la sottrazione di macchina non introduce alcun errore) ha la sua origine negli errori presenti nei due operandi; o meglio, l'operazione di sottrazione ha amplificato detti errori. Se i due operandi sono privi di errori, il risultato non presenta alcuna perdita di precisione, e gli zeri finali aggiunti sono esatti.

Stabilità di un algoritmo

Spesso un errore iniziale si propaga in una sequenza di calcoli. Una qualità desiderata di ogni processo di calcolo numerico è che un piccolo errore iniziale produca piccole variazioni nel risultato finale. Un algoritmo che presenta questa proprietà si dice *stabile*; altrimenti, si dice *instabile*. Laddove è possibile saranno, ovviamente, preferiti metodi stabili. In altre parole, se si introduce un errore ad un certo passo (j) come si ripercuote al passo successivo $(j+1)$? Se l'errore cresce il procedimento sarà instabile viceversa sarà stabile. Occorrerà dunque verificare che

$$|\epsilon^{(j+1)}| < |\epsilon^{(j)}|$$

Osserviamo inoltre, che la stabilità può essere condizionata o incondizionata a seconda che dipenda da opportune condizioni sui parametri del problema/algoritmo, oppure no.

Quantitativamente, la propagazione dell'errore può essere descritta mediante la seguente definizione.

Definizione. Supponiamo che $E(n)$ rappresenti la crescita dell'errore dopo n passi. Se $|E(n)| \approx n\epsilon$, la crescita dell'errore si dice *lineare*. Se $|E(n)| \approx K^n\epsilon$, la crescita dell'errore si dice *esponenziale*. Se $K > 1$, l'errore esponenziale cresce indefinitamente per $n \to \infty$, e se $0 < K < 1$, l'errore esponenziale decresce a zero per $n \to \infty$.

1.2.4 Errori di troncamento

Gli errori di troncamento derivano dall'uso di un procedimento di approssimazione in sostituzione di operazioni matematiche esatte. Per comprendere le caratteristiche di tali errori, prendiamo in esame una formula matematica molto spesso utilizzata nei metodi numerici per esprimere una funzione in maniera approssimata: la serie di Taylor.

Serie di Taylor

Una serie di Taylor a infiniti termini può essere utilizzata per calcolare il valore di una funzione in corrispondenza di un dato valore della variabile indipendente x. Analogamente, la serie di Taylor permette di valutare il valore di una funzione nel punto x_{i+1} in termini del valore della funzione e delle sue derivate in un vicino punto x_i.

Invece di presentare immediatamente la serie di Taylor nella sua completezza, la costruiremo termine per termine allo scopo di comprenderne meglio il comportamento. Considerando solo il primo termine della serie, si ha

$$f(x_{i+1}) \simeq f(x_i) \tag{1.6}$$

Questa relazione, che viene detta *approssimazione di ordine zero*, indica che il valore di f nel nuovo punto x_{i+1} è lo stesso che aveva nel punto precedente x_i. Questo risultato ha senso, anche intuitivamente, in quanto, se x_i e x_{i+1} sono vicini, è probabile che i corrispondenti valori di f non differiscano di molto.

Ovviamente, se la funzione che viene approssimata è una costante, allora la (1.6) fornisce una stima esatta. Se, però, la funzione varia lungo tutto l'intervallo di interesse, sono necessari altri termini della serie di Taylor per ottenere una stima migliore. Per esempio, la *approssimazione del primo ordine* si ottiene aggiungendo il secondo termine alla serie:

$$f(x_{i+1}) \simeq f(x_i) + f'(x_i)(x_{i+1} - x_i) \tag{1.7}$$

Il termine del primo ordine è costituito da una pendenza $f'(x_i)$ moltiplicata per la distanza tra x_i e x_{i+1}; ora, quindi, l'espressione rappresenta una linea retta di pendenza variabile ed è in grado di approssimare un aumento o una diminuzione della funzione tra x_i e x_{i+1}.

Anche se la (1.7) è in grado di tener conto di una variazione, l'approssimazione che essa dà è valida solo se questa variazione è *lineare*, cioè se può essere rappresentata da una retta. Aggiungiamo alla serie, quindi, un altro termine, ottenendo l'approssimazione del *secondo ordine* che ci permetta, almeno in parte, di tener conto della eventuale curvatura della funzione:

$$f(x_{i+1}) \simeq f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2!}(x_{i+1} - x_i)^2 \tag{1.8}$$

Continuando ad aggiungere i termini, si arriva ad ottenere lo sviluppo completo in serie di Taylor.

$$\begin{aligned} f(x_{i+1}) &= f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2!}(x_{i+1} - x_i)^2 + \frac{f'''(x_i)}{3!}(x_{i+1} - x_i)^3 + \dots \\ &+ \frac{f^{(n)}(x_i)}{n!}(x_{i+1} - x_i)^n + R_n \end{aligned} \tag{1.9}$$

Siccome la (1.9) è una serie infinita, il segno di approssimazione usato dalla (1.6) alla (1.8) può essere sostituito da un segno di uguale. L'espressione contiene un termine *resto* R_n che tiene conto di tutti i termini da $n+1$ all'infinito:

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x_{i+1} - x_i)^{n+1} \tag{1.10}$$

dove n indica che il resto viene calcolato per l'approssimazione di ordine n-esimo e ξ è un valore compreso tra x_i e x_{i+1}.

Tale resto di R_n è il valore dell'errore di troncamento che si ha se la serie viene troncata all'n-esimo ordine.

Conviene spesso semplificare l'espressione della serie di Taylor introducendo il passo $h = x_{i+1} - x_i$ ed esprimendo la (1.9) come

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f'''(x_i)}{3!}h^3 + \dots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n \tag{1.11}$$

dove

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1} \tag{1.12}$$

In generale, lo sviluppo in serie di Taylor di ordine n-esimo risulta esatto per un polinomio di ordine n. Nel caso di altre funzioni continue e derivabili, come la sinusoide o la funzione esponenziale, un numero finito di termini probabilmente non è in grado di dare una stima esatta. Ciascun nuovo termine contribuirà a migliorare, per quanto poco, l'approssimazione: la serie dà un risultato esatto solo se viene sommato un numero infinito di termini.

Nonostante quanto appena detto, lo sviluppo in serie di Taylor è utile in pratica in quanto, nella maggior parte dei casi, prendendo in considerazione solo pochi termini si ottiene un'approssimazione abbastanza vicina al valore reale per gli scopi pratici. La determinazione del numero di termini necessari per ottenere un'approssimazione "abbastanza buona" si basa sulla valutazione del resto dello sviluppo la cui forma generale, ricordiamo, è data dalla (1.12).

Questa relazione comporta principalmente due svantaggi: in primo luogo, ξ non è noto, esattamente ma si sa soltanto che si trova da qualche parte tra x_i e x_{i+1}. Inoltre, per utilizzare la (1.12) bisogna determinare la derivata $(n+1)$-esima di $f(x)$ e, per far ciò, bisogna conoscere $f(x)$. Se conoscessimo $f(x)$, però, non avremmo alcun bisogno di sviluppare la funzione in serie di Taylor!

Nonostante questo dilemma, la (1.12) è ancora utile in quanto ci permette di approfondire la nostra conoscenza sugli errori di troncamento; ciò è possibile in quanto, in effetti, siamo in grado di controllare il termine h^{n+1} nell'equazione. In altre parole, possiamo decidere a che distanza da x_i vogliamo valutare $f(x)$ e quanti termini includere nello sviluppo. Di conseguenza, la (1.12) viene solitamente espressa come

$$R_n = O(h^{n+1})$$

nella quale la notazione $O(h^{n+1})$ significa che l'errore di troncamento è dell'ordine di h^{n+1}. L'errore, cioè, è proporzionale al passo h elevato alla $n+1$. Anche se questa approssimazione non dice nulla riguardo all'ordine di grandezza delle derivate che moltiplicano h^{n+1}, essa è estremamente utile per apprezzare l'errore relativo di metodi numerici basati sullo sviluppo in serie di Taylor. Per esempio, se l'errore è $O(h)$, un dimezzamento dell'ampiezza del passo dimezzerà anche l'errore, mentre, se l'errore è $O(h^2)$, un passo grande la metà dà un quarto dell'errore di partenza. Tutto ciò è vero solo se la derivata in ξ è quasi costante. In generale, possiamo contare sul fatto che aumentando il numero di termini della serie di Taylor l'errore di troncamento diminuisce. Inoltre, se h è sufficientemente piccolo, il primo termine più pochi altri costituiscono solitamente l'origine di una percentuale sproporzionatamente grande dell'errore; pertanto, sono sufficienti solo alcuni termini per ottenere una stima adeguata.

Resto dell'espansione in serie di Taylor

Spieghiamo come mai è stato introdotto il parametro ξ nella (1.12). Invece di presentare una rigorosa derivazione matematica, svilupperemo una descrizione più semplice basata su di una interpretazione di tipo grafico; in seguito, estenderemo questo caso particolare in modo da ricondurci alla formulazione generale.

Supponiamo di troncare lo sviluppo in serie di Taylor (1.11) al termine di ordine zero per ottenere

$$f(x_{i+1}) \simeq f(x_i)$$

La rappresentazione grafica di questa approssimazione di ordine zero è mostrata in Fig. 1.4a dove è mostrato anche il resto, o errore, dell'approssimazione, che è costituito dalla serie infinita di termini che sono stati trascurati:

$$R_0 = f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f'''(x_i)}{3!}h^3 + \ldots$$

Evidentemente, non è molto comodo valutare il resto quando questo è sotto forma di serie infinita. Una soluzione consiste nel troncare anche il resto al primo termine:

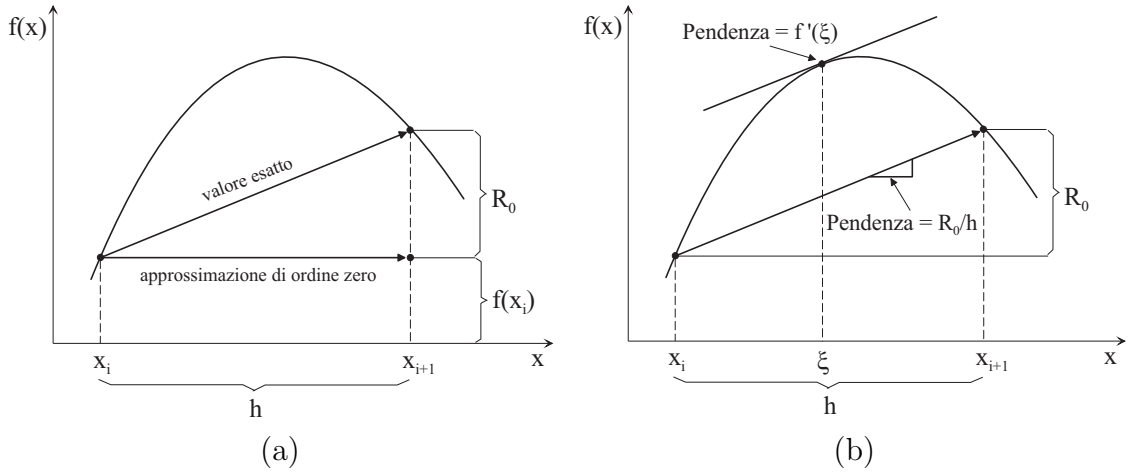


Figura 1.4: (a) Rappresentazione grafica di uno sviluppo in serie di Taylor di ordine zero e del relativo resto; (b) rappresentazione grafica del teorema del valor medio

$$R_0 \simeq f'(x_i)h \tag{1.13}$$

Come è stato detto in precedenza, le derivate di ordine inferiore rappresentano il contributo maggiore nella costituzione del resto; ciò nonostante, la semplificazione rappresentata dalla (1.13) è ancora inesatta a causa del termine di secondo grado e dei termini superiori. Questa mancanza di esattezza viene indicata dall'apposito simbolo di uguaglianza approssimativa ($\simeq$) inserito nella (1.13).

Per trasformare l'approssimazione in una equivalenza facciamo ricorso a una rappresentazione grafica: nella Fig. 1.4b sarebbe possibile determinare l'errore R_0 se la posizione del valore esatto fosse nota. Ovviamente questo valore non è noto: in caso contrario, non ci sarebbe alcun bisogno di calcolare la serie di Taylor. Esiste, però, un teorema che permette di ridefinire il problema in modo da aggirare in parte il problema.

Il *teorema del valor medio* stabilisce che se una funzione $f(x)$ e la sua derivata prima sono continue in un intervallo $[x_i, x_{i+1}]$, allora esiste almeno un punto $(\xi, f(\xi))$ della funzione, con $x_i < \xi < x_{i+1}$, tale che la tangente della funzione in quel punto è parallela alla linea che congiunge $f(x_i)$ e $f(x_{i+1})$ (v. Fig. 1.4b). Come esempio pratico di applicazione di questo teorema immaginiamo di muoverci a velocità variabile tra due punti: intuitivamente, ci sarà almeno un istante in cui la velocità istantanea è pari al valore medio della velocità.

Con l'aiuto di questo teorema, risulta evidente che, come illustrato in Fig. 1.4b, la pendenza $f'(\xi)$ è uguale all'incremento della funzione R_0 diviso per l'incremento h della variabile indipendente:

$$f'(\xi) = \frac{R_0}{h}$$

Questa espressione può anche essere così formulata:

$$R_0 = f'(\xi)h \tag{1.14}$$

Abbiamo così derivato la versione di ordine zero della (1.12). Le versioni di ordine superiore non sono altro che un'estensione logica del procedimento usato per ricavare la (1.14), basato sulla forma generale del teorema del valor medio esteso (Thomas e Finney, 1979). La versione del primo ordine, perciò, è:

$$R_1 = \frac{f''(\xi)}{2!}h^2 \tag{1.15}$$

In questo caso, il valore assunto da ξ è tale per cui la derivata seconda soddisfa la (1.15). Dalla (1.12) è possibile ricavare allo stesso modo le versioni di ordine superiore.

Molti dei metodi numerici presentati nel seguito descrivono tecniche di rappresentazione di complesse espressioni matematiche per mezzo di approssimazioni più semplici: per la sua capacità di scomporre un'espressione nelle sue componenti di ordine inferiore e superiore, lo sviluppo in serie di Taylor si rivelerà di grande utilità come strumento di analisi e comprensione dei metodi numerici.

1.2.5 Errore numerico globale

L'errore numerico globale è la somma degli errori di troncamento e di arrotondamento. Spesso, l'unico modo di ridurre gli errori di arrotondamento consiste nell'aumentare il numero di cifre significative trattabili dal calcolatore. Inoltre, gli errori di arrotondamento tendono ad aumentare al crescere del numero di operazioni richieste da un'elaborazione. Ad esempio, le stime della derivata migliorano al diminuire del passo; dato che un passo più piccolo comporta un numero maggiore di operazioni, si ha che l'errore di troncamento decresce al crescere del numero di operazioni. In conclusione, abbiamo di fronte il seguente dilemma: l'espediente che permette di diminuire una componente dell'errore globale porta ad un aumento dell'altra componente. In generale, si tende a diminuire il passo per minimizzare gli errori di troncamento, per poi scoprire che, così facendo, l'errore di arrotondamento tende a dominare la soluzione e l'errore globale cresce: succede così che il rimedio è peggiore del male (Fig. 1.5).

Ciò di cui abbiamo bisogno è un criterio che ci permetta di scegliere un passo abbastanza ampio, tale da diminuire il numero di operazioni necessarie e da minimizzare gli errori di arrotondamento, ma piccolo abbastanza da evitare gli errori di troncamento. Se l'errore globale ha l'andamento visualizzato in Fig. 1.5, l'ideale sarebbe conoscere la posizione del punto nel quale gli errori di arrotondamento cominciano ad annullare i benefici di una riduzione del passo.

Nei casi comuni, però, situazioni come quella descritta sono abbastanza rare, dato che in generale i calcolatori dispongono di un numero di cifre significative tale da annullare l'importanza degli errori di arrotondamento. Ciò nonostante, tali situazioni si presentano effettivamente, suggerendo una specie di "principio di indeterminazione numerica" che

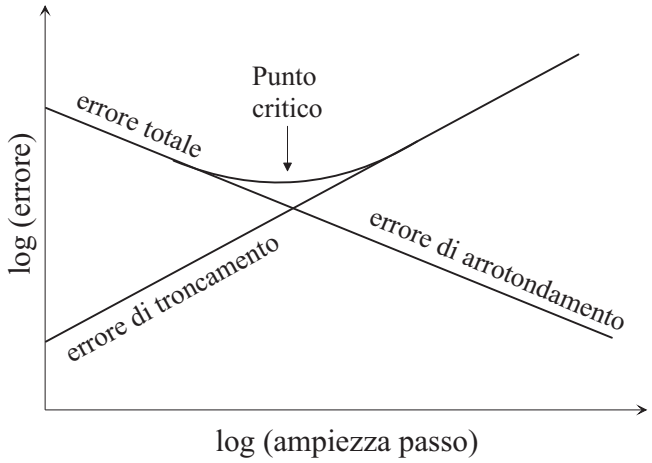


Figura 1.5: Rappresentazione grafica del compromesso tra l'errore di arrotondamento e l'errore di troncamento che talvolta si presenta utilizzando un dato metodo numerico. Viene indicato il punto di minimo, nel quale gli arrotondamenti cominciano ad annullare il beneficio ottenuto con la riduzione del passo.

pone un limite invalicabile all'accuratezza ottenibile utilizzando certi metodi numerici su calcolatore.

A causa di questi limiti, la nostra capacità di valutare gli errori risulta limitata; come conseguenza, la valutazione degli errori nei metodi numerici diventa una specie di arte che dipende in parte dai suggerimenti ottenibili da prove pratiche e in parte dall'intuito e dall'esperienza dell'utilizzatore.

1.2.6 Condizionamento di un problema

Per valutare la bontà della risposta fornita da un algoritmo quando viene utilizzato per la risoluzione di un problema numerico $y = f(x)$, è necessario conoscere anche la "reazione" di quest'ultimo all'introduzione di *perturbazioni* nei dati iniziali x. Occorre cioè sapere come gli errori (inevitabili) presenti nei dati vengono propagati dal problema (la funzione f), in assenza di errori di calcolo e di rappresentazione dei numeri. Questo studio ci consente di stabilire la massima precisione raggiungibile da un "buon" algoritmo (implementato su di un elaboratore, e quindi operante con l'aritmetica di macchina). Infatti, poiché in generale non avremo i dati iniziali x, ma una loro approssimazione $\bar{x} = x(1 + d)$, per esempio $\bar{x} = \mathrm{fl}(x)$, la funzione f come risposta ci fornirà $f(\bar{x})$. Partendo con i dati $\bar{x}$ l'obiettivo dell'algoritmo diventa $f(\bar{x})$ e non più $f(x)$.

Pertanto, per giudicare la bontà di un algoritmo proposto per il calcolo di $f(x)$ dobbiamo confrontare la risposta y^*, fornita dall'algoritmo, con $f(\bar{x})$. Diremo che l'algoritmo proposto

è stabile quando la quantità

$$\frac{\|f(\bar{x}) - y^*\|}{\|f(\bar{x})\|}, \qquad f(\bar{x}) \neq 0,$$

è dell'ordine di grandezza della precisione di macchina ε_m, e instabile altrimenti.

L'esame dell'errore relativo

$$\frac{\|f(x) - f(\bar{x})\|}{\|f(x)\|}, \qquad f(x) \neq 0,$$

quale funzione delle perturbazioni relative $\frac{\|dx\|}{\|x\|}$ ($x \neq 0$), introdotte nei dati iniziali x, conduce invece alla definizione del concetto di *condizionamento del problema*. Quando a piccole perturabazioni (relative) dei dati x corrispondono perturbazioni (relative) su $f(x)$ dello stesso ordine di grandezza, il problema $y = f(x)$ è definito *ben condizionato*; altrimenti è detto *mal condizionato*. Ovviamente, il condizionamento del problema è una funzione dei dati x.

Il condizionamento è proprio del problema numerico (o della funzione f), e non ha alcun legame né con gli errori di arrotondamento delle operazioni di macchina né con il particolare procedimento di calcolo seguito per determinare la risposta $f(x)$. Solo quando il problema è ben condizionato, e l'algoritmo proposto per la sua risoluzione è stabile, gli errori (relativi) introdotti nei dati e nelle operazioni di macchina non vengono amplificati sul risultato.

Vi sono problemi per i quali esiste la soluzione ed esiste un algoritmo per calcolarla, ma che sono praticamente irrisolubili perché fortemente malcondizionati, per cui i soli errori di rappresentazione dei dati generano sul risultato errori superiori al 100%.

Esempio. Si consideri il seguente sistema di equazioni lineari

$$\begin{cases} x + y = 2 \\ 1001x + 1000y = 2001 \end{cases}$$

che ha la soluzione $x = 1$, $y = 1$. Si alteri il coefficiente della x, nella prima equazione, dell'1%, e si consideri il nuovo sistema perturbato

$$\begin{cases} (1 + 1/100)x + y = 2 \\ 1001x + 1000y = 2001 \end{cases}$$

che ha soluzione $\tilde{x} = -1/9$, $\tilde{y} = 1901/900$. La soluzione del sistema perturbato presenta, rispetto alla soluzione del sistema non perturbato, una variazione maggiore del 110% sia nella x che nella y.

La natura del mal condizionamento del problema precedente può essere descritta mediante un'interpretazione geometrica. Le due equazioni del sistema corrispondono alle de rette tracciate con linea continua nella Fig. 5.3 e la loro intersezione ha per componenti

[3] $\|\cdot\|$ denota una norma di vettore; vedi Appendice C.

la soluzione del sistema. Poiché le due rette formano tra loro un angolo molto piccolo, perturbando la prima equazione, si altera di poco la posizione della prima retta (ottenendo la retta tratteggiata), ma cambia molto la posizione del punto intersezione e quindi le coordinate di questo che sono le soluzioni del sistema perturbato. Un modo per misurare il

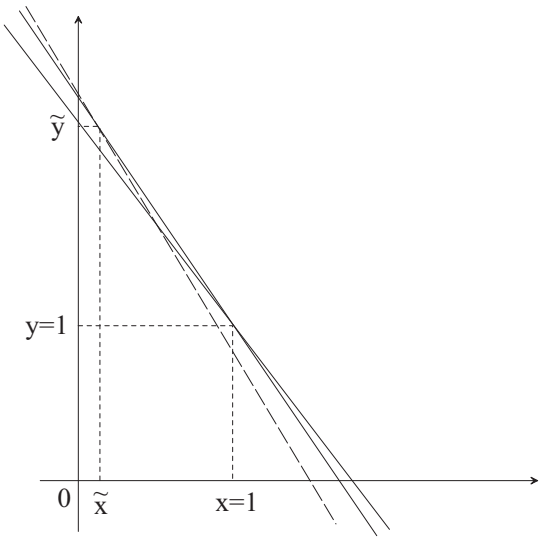


Figura 1.6: Interpretazione geometrica del mal condizionamento di un sistema lineare

condizionamento nel caso in cui il problema sia quello di calcolare il valore che una funzione f di una variabile reale assume in un punto $x \neq 0$, consiste nel valutare la variazione che si produce nel valore della funzione quando il dato x viene perturbato con una perturbazione relativa d. La corrispondente variazione relativa del risultato è

$$r = \frac{f(x(1+d)) - f(x)}{f(x)}$$

e il condizionamento viene misurato per mezzo del quoziente r/d. Se la funzione $f(x)$ è derivabile, si ha

$$\lim_{d \to 0} = \frac{r}{d} = \frac{x f'(x)}{f(x)} \ .$$

Tale quantità, che non dipende dalla perturbazione d, dà un indicazione di quanto è amplificato l'errore introdotto nella variabile indipendente.

Nel caso di malcondizionamento, per trattare il problema conviene aumentare la precisione della rappresentazione dei dati, aumentando il numero di cifre, in modo da ridurre l'errore di rappresentazione. E' per questo che su molti calcolatori è possibile utilizzare, oltre alla precisione semplice, in cui la rappresentazione in base è fatta con un numero standard di cifre, anche la *precisione doppia*, in cui la rappresentazione in base avviene con un numero all'incirca doppio di cifre, e più in generale, la *precisione multipla*, in cui si usa un numero arbitrario di cifre. In alcuni casi, se l'errore numerico cresce esponenzialmente con la dimensione del problema, l'aumento del numero di cifre non fornisce alcun concreto

vantaggio nella riduzione dell'errore. Esistono comunque molti casi in cui questo modo di operare permette di contenere l'errore.

Si presentano però due inconvenienti: per memorizzare un numero occorre una maggior quantità di memoria, con conseguente riduzione delle dimensioni massime dei problemi trattabili; il tempo necessario per eseguire un'operazione su due numeri aumenta all'aumentare delle cifre, con conseguente notevole incremento del tempo di elaborazione. Sul calcolatore una moltiplicazione viene eseguita con un metodo analogo a quello usato nel calcolo "con carta e penna" di un prodotto di due numeri di n cifre decimali che richiede n^2 moltiplicazioni di numeri di una cifra. Quindi raddoppiando il numero di cifre risulta quadruplicato il tempo richiesto per ogni moltiplicazione: se ad esempio un personal computer esegue mille moltiplicazioni al secondo, raddoppiando il numero di cifre eseguirà solamente 250 moltiplicazioni al secondo. L'uso di un numero maggiore di cifre per ridurre la propagazione degli errori comporta quindi un impiego maggiore delle risorse *quantità di memoria* e *tempo di elaborazione*.

Bibliografia

G. Monegato, *Fondamenti di calcolo numerico*, Libreria Editrice Universitaria Levrotto&Bella, Torino, 1990

S.C. Chapra, R.P. Canale, *Metodi numerici per l'ingegneria*, McGraw-Hill Italia, 1988

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

J.H. Mathews, *Numerical methods*, Prentice Hall (2^a ed.), 1992

O. Caligaris, P. Oliva, *Analis Matematica I*, ECIG Genova (2^a ed.), 1986

S. Ceri, D. Mandrioli, L. Sbattella, *Informatica - Arte e mestiere*, McGraw-Hill Italia, 1999

P. Tosoratti, *Introduzione all'informatica*, Casa Editrice Ambrosiana (2^a ed.), 1998

R. Bevilacqua, D.B.M. Capovani, O. Menchi, *Metodi numerici*, Zanichelli, 1992

Capitolo 2

Integrazione e derivazione di una funzione

2.1 Integrazione

Tema principale di questo capitolo è la valutazione numerica di integrali definiti

$$I(f) = \int_a^b f(x)dx \ .$$

Spesso è impossibile determinare $I(f)$ per via analitica; ma anche quando tale via risulta percorribile, l'espressione finale è sovente così complessa rispetto alla funzione integranda f da suggerire l'uso di approcci più semplici. Inoltre l'eventuale soluzione analitica potrebbe coinvolgere funzioni elementari e non, che devono poi venire valutate (e quindi approssimate). Se invece la funzione $f(x)$ è nota solo per punti, oppure è valutabile per ogni valore dell'argomento x mediante una *routine*, l'approccio analitico non può neppure essere preso in considerazione. Pertanto, supponendo di conoscere o di poter valutare la funzione integranda $f(x)$ nei punti $\{x_i\} \subseteq [a, b]$, prefissati oppure da noi scelti, esaminiamo alcuni procedimenti per il calcolo approssimato del suo integrale definito tra gli estremi a e b.

2.1.1 Integrale come somma

Intuitivamente il valore dell'integrale definito di una funzione può essere considerato come l'area sottesa dalla funzione stessa rispetto all'asse coordinato. E' possibile formulare un metodo approssimato che sfrutta direttamente questa proprietà.

Richiamiamo alcuni concetti preliminari sull'integrazione di funzioni.

Sia $f(x)$ definita in $[a, b]$ e ivi limitata. Chiamiamo partizione di $[a, b]$ un insieme $P = \{x_0, x_1, \ldots, x_{n-1}\}$ di punti di $[a, b]$ tali che

$$a = x_0 < x_1 < \cdots < x_{n-1} = b \ .$$

Se definiamo

$$m_i = \inf\{f(x), x \in [x_i, x_{i+1}]\}\ , \qquad M_i = \sup\{f(x), x \in [x_i, x_{i+1}]\}$$

allora

$$s(f,P) = \sum_{i=0}^{n-1} m_i(x_{i+1} - x_i) \leq \int_a^b f(x)dx \leq \sum_{i=0}^{n-1} M_i(x_{i+1} - x_i) = S(f,P)$$

dove $s(f,P)$ e $S(f,P)$ sono dette, rispettivamente, *somma inferiore* e *somma superiore* di f rispetto alla partizione P[1]. Mediante tali somme il valore dell'integrale è approssimato per difetto o per eccesso. In particolare, se viene scelta la partizione

$$P_n = \left\{ a + i\frac{(b-a)}{n-1}, i = 0, 1, \ldots, n-1 \right\}$$

si ha che

$$\lim_{n\to\infty} s(f,P_n) = \sup\{s(f,P_n), n \in N\} = \int_a^b f(x)dx = \inf\{S(f,P_n), n \in N\} = \lim_{n\to\infty} S(f,P_n)$$

Pertanto, possiamo direttamente utilizzare le definizioni di $s(f,P)$ e $S(f,P)$ per una valutazione approssimata dell'integrale I, inoltre per $n \to \infty$ l'approssimazione tende al valore esatto assicurandoci la convergenza del procedimento (v. Fig. 2.1).

2.1.2 Metodo di calcolo dell'integrale per somme di rettangoli

Consideriamo n sottointervalli di $[a,b]$ di ampiezza $h = \frac{(b-a)}{n-1}$, dove h è detto *passo di integrazione*. In queste ipotesi $[x_i, x_{i+1}] = [x_i, x_i + h]$. Possiamo applicare direttamente le definizioni di somme inferiori e superiori e approssimare così il valore dell'integrale I come media aritmetica delle sue approssimazioni attraverso le somme inferiori e superiori:

```
h=(b-a)/((float) (n-1));
x=a;
y=f(a);
sum=0.;
SUM=0.;
for(i=0;i<n;i++)
 {
   yprec=y;
   x+=h;
   y=f(x);
   sum+=h*min(y,yprec);
```

[1]Si suppone di conoscere i valori della $f(x)$ solo nei punti della partizione

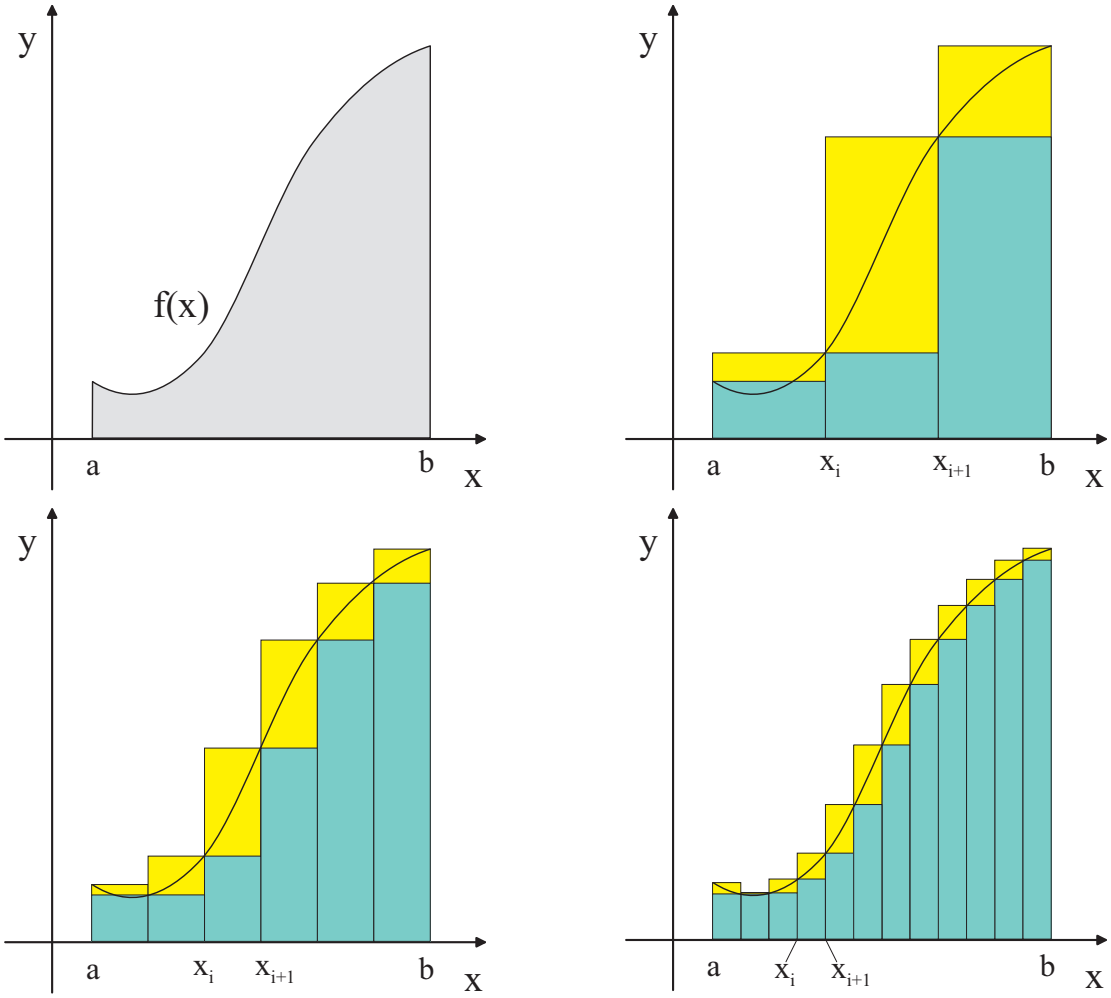


Figura 2.1: Rappresentazione grafica del calcolo approssimato di un integrale definito mediante somme superiori e somme inferiori, per un numero crescente dei punti della partizione

```
   SUM+=h*max(y,yprec);
 }
I_inf=sum;
I_sup=SUM;
I=0.5*(I_inf+I_sup);
```

dove `sum`$\longrightarrow s(f,P_n)$ e `SUM`$\longrightarrow S(f,P_n)$ e dove

$$\min(p,q)=\begin{cases} p & \text{se } p<q \\ q & \text{altrimenti} \end{cases} \qquad \max(p,q)=\begin{cases} p & \text{se } p\geq q \\ q & \text{altrimenti} \end{cases}$$

Considerato che, al crescere di n, sia le somme inferiori sia quelle superiori convergono al valore esatto dell'integrale, le prime attraverso approssimazioni per difetto, le seconde attraverso approssimazioni per eccesso, è anche possibile costruire una formula approssimata (*formula di Eulero*) che evita di valutare la condizione sul valore massimo e minimo della funzione rispetto agli estremi del sottointervallo $\{x_i, x_i+h\}$

$$I_h(f)=\sum_{i=0}^{n-1} f(x_i)(x_{i+1}-x_i)=h\sum_{i=0}^{n-1} f(x_i) \tag{2.1}$$

Il codice per il calcolo dell'integrale può quindi essere semplificato nel modo seguente:

```
h=(b-a)/((float) (n-1));
x=a;
I=0.;
for(i=0;i<n;i++)
 {
   y=f(x);
   I+=h*y;
   x+=h;
 }
```

Esercizi proposti

1. Scrivere un programma per la valutazione dell'integrale attraverso le somme inferiori e superiori.

2. Provare a ridurre progressivamente il passo di integrazione, osservando le conseguenze sulla precisione del calcolo quando h si avvicina alla precisione di macchina ε_m.

2.1.3 Metodi interpolatori

I metodi di integrazione numerica sin qui presentati sfruttano direttamente la definizione di integrabilità di una funzione nell'intervallo $[a,b]$. Più in generale, il calcolo numerico

fornisce numerose **formule**, dette **di quadratura**, del tipo

$$Q_n(f) = \sum_{i=0}^{n-1} w_i f(x_i) \tag{2.2}$$

tali che

$$I(f) = \int_a^b f(x)dx \approx Q_n(f) \ .$$

I numeri (reali) $\{x_i\} \subseteq [a, b]$ e $\{w_i\}$ vengono chiamati rispettivamente *nodi* e *pesi* della formula di quadratura. L'idea alla base dei metodi interpolatori è quella di scegliere i pesi della formula di quadratura sulla base di un'interpolazione dei valori della f con funzioni elementari. In particolare, la funzione integranda $f(x)$ viene approssimata con un polinomio di $L_n(f; x)$, di grado $k \leq n-1$, unico, che interpola la funzione nei nodi $\{x_i\}$. I coefficienti (pesi) nella formula di quadratura sono ricavati calcolando analiticamente il valore dell'integrale del polinomio interpolatore che passa per $k+1$ punti della $f(x)$. Le formule costruite in questo modo vengono chiamate interpolatorie e risultano "esatte" ogniqualvolta $f(x)$ è un polinomio di grado minore o uguale a k.

Affinchè la formula di quadratura (2.2) definisca una "buona" discretizzazione dell'integrale è necessario che

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} w_i f(x_i) = \int_a^b f(x)dx$$

in questo caso diciamo che la formula è convergente. Più regolare è la funzione $f(x)$ più rapida è la convergenza della quadratura al valore esatto dell'integrale.

Si può osservare che la formula di Eulero, detta anche formula di quadratura rettangolare, presentata nel paragrafo precedente, si ottiene quando il grado del polinomio $k = 0$; le formule derivanti da polinomi interpolatori di grado $k > 0$ consentono, in generale, di ottenere, a parità di tempo di calcolo, precisioni migliori; in particolare, largamente utilizzate sono la *formula dei trapezi* o di Bezout, ottenuta per $k = 1$ e quella di *Simpson*, ottenuta per $k = 2$.

Formula di Eulero (k=0) Detto f_0 il valore di $f(x)$ nel punto x_0, la formula di Eulero deriva dall'integrazione della costante f_0 nell'intervallo $[x_0, x_0 + h]$

$$\int_{x_0}^{x_0+h} f(x)dx \approx hf_0 \tag{2.3}$$

Estendendo la (2.3) a tutto l'intervallo $[a, b]$ si ottiene:

$$\int_{x_0}^{x_{n-1}} f(x)dx \approx Q_n(f) = \ h(f_0 + f_1 + \cdots + f_{n-1}) \tag{2.4}$$

Formula dei trapezi (k=1) Detti f_0 e f_1 i valori di $f(x)$ nei punti x_0 e $x_0 + h$, la formula dei trapezi deriva dall'integrazione della retta che passa per i punti f_0 e f_1

$$\int_{x_0}^{x_0+h} f(x)dx \approx \frac{h}{2}(f_0 + f_1) \tag{2.5}$$

Estendendo la (2.5) a tutto l'intervallo $[a, b]$:

$$\int_{x_0}^{x_{n-1}} f(x)dx \approx Q_n(f) = h\left(\frac{1}{2}f_0 + f_1 + f_2 + \cdots + f_{n-2} + \frac{1}{2}f_{n-1}\right) \tag{2.6}$$

Formula di Simpson (k=2) Detti f_0, f_1 e f_2 i valori di $f(x)$ nei punti x_0, $x_0 + h$ e $x_0 + 2h$, la formula di Simpson deriva dall'integrazione della parabola che passa per i punti f_0, f_1, f_2:

$$\int_{x_0}^{x_0+2h} f(x)dx \approx \frac{h}{3}(f_0 + 4f_1 + f_2) \tag{2.7}$$

Estendendo la formula all'intero intervallo $[a, b]$ si ottiene:

$$\int_{x_0}^{x_{n-1}} f(x)dx \approx Q_n(f) = \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-3} + 4f_{n-2} + f_{n-1}\right) \tag{2.8}$$

2.1.4 Stima dell'errore

Supponiamo di aver scelto una formula di quadratura

$$Q_n(f) = \sum_{i=0}^{n} w_i f(x_i)$$

per approssimare l'integrale

$$I(f) = \int_a^b f(x)dx\ .$$

Affinchè la stima $Q_n(f)$ sia credibile occorre poter associare ad essa un'indicazione della sua precisione, valutando lo scarto

$$R_n(f) = I(f) - Q_n(f)$$

Per le formule di tipo interpolatorio si può dare una rappresentazione integrale dell'errore:

$$R_n(f) = \int_a^b E_n(f; x)dx \qquad \text{dove} \qquad E_n(f; x) = f(x) - L_n(f; x)$$

altre sono reperibili nella letteratura specializzata. Tuttavia, queste espressioni dell'errore hanno un'importanza soprattutto teorica. Solitamente, la stima dell'errore $R_n(f)$ viene

prodotta nel modo seguente: si prende una seconda formula $Q_m(f)$ dello stesso tipo di $Q_n(f)$, ma più "precisa", ovvero con un numero maggiore di nodi ($m > n$), e si pone

$$|R_n(f)| \approx |Q_n(f) - Q_m(f)| \, .$$

Generalmente si sceglie $m = n + 1$ quando la funzione $f(x)$ è ritenuta sufficientemente "regolare", e $m = 2n$ quando non lo è. In pratica, allo scopo di eseguire il calcolo dell'integrale $I(f)$ con precisione ε preassegnata si può iterativamente valutare l'integrale mediante una formula di quadratura con valori decrescenti del passo di integrazione h. Ad esempio, dimezzando iterativamente h (e quindi raddoppiando i nodi, vedi Fig. 2.2) e arrestando il procedimento di calcolo quando risulta

$$|Q_n(f) - Q_{2n}(f)| < \varepsilon$$

Si osserva che per quanto detto sulla precisione delle operazioni di macchina, non potremo considerare passi di intregrazione $h < \varepsilon_m$.

E' inoltre immediato verificare che i metodi di integrazione numerica qui presentati risultano stabili e ben condizionati.

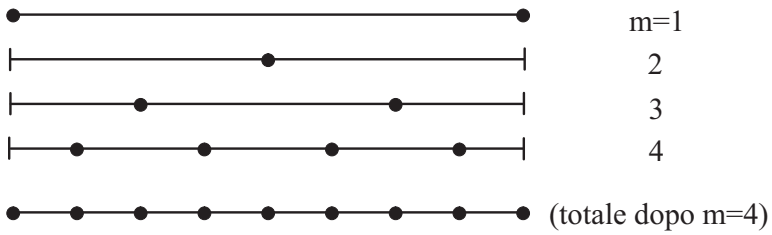


Figura 2.2: Rappresentazione schematica dei punti dell'intervallo ove occorre valutare la funzione considerando suddivisioni successive del passo di integrazione. Si osservi che ad ogni suddivisione rimangono validi i punti calcolati precedentemente e devono essere calcolati solo i nuovi punti

Esercizi proposti

1. Scrivere un programma che calcoli il valore approssimato dell'integrale definito di una funzione predefinita utilizzando la formula dei trapezi. Valutare la precisione dell'approssimazione al diminuire del passo di integrazione h.

2. Modificare il programma dell'esercizio 1 in modo che possa essere valutato l'integrale con una precisione ε preassegnata.

3. Scrivere un programma che calcoli il valore approssimato dell'integrale definito di una funzione predefinita utilizzando il metodo di Simpson.
   Suggerimento: Si osservi che la formula di Simpson può essere riscritta come:

$$\frac{h}{3}(S1 + 2S2_h + 4S4_h)$$

dove $S1 = f_0 + f_{n-1} = f(a) + f(b)$, $S2_h = f_2 + f_4 + \cdots + f_{n-3}$, $S4_h = f_1 + f_3 + \cdots + f_{n-2}$.

4. Provare a modificare il programma dell'esercizio 3 utilizzando una procedura basata sulla formula dei trapezi.
   Suggerimento: Le formule dei trapezi ottenute iterativamente dalla suddivione del passo di integrazione definiscono una successione del tipo:

$$Q_1(f) = \left(\frac{1}{2}f_0 + \frac{1}{2}f_N\right) h_1 \quad f_0 = f(a), \quad f_N = f(b), \quad h_1 = (b-a), \quad N = 2^M, M \text{ intero}$$

$$Q_{2n}(f) = \frac{1}{2}Q_n(f) + (f_{N/2n} + f_{3N/2n} + \cdots + f_{(2n-1)N/2n})h_{2n}, \quad h_{2n} = \frac{h_n}{2}, \quad n = 1, \ldots, \frac{N}{2}$$

   con N numero massimo di sottointervalli in cui è suddiviso l'intervallo (a, b). In queste ipotesi si osserva che la formula di Simpson si può scrivere come

$$\frac{4}{3}Q_{2n}(f) - \frac{1}{3}Q_n(f) \, .$$

2.1.5 Formule su intervalli aperti e semi-aperti

Le formule presentate permettono di valutare l'integrale nell'intero intervallo $[a, b]$. Tali formule, poiché usano i valori della funzione agli estremi ($f(a)$, $f(b)$) sono dette *formule chiuse*. Occasionalmente, ci potrebbe capitare di voler integrare una funzione per cui sia difficile valutarne il valore in uno o entrambi gli estremi (ad es., la valutazione di f comporta un limite del tipo "zero su zero", o peggio, la funzione f presenta una singolarità integrabile). In questi casi dobbiamo ricorrere a *formule aperte*, che stimano l'integrale usando solo i nodi x_i strettamente compresi in (a, b) (vedi Fig. 2.3).

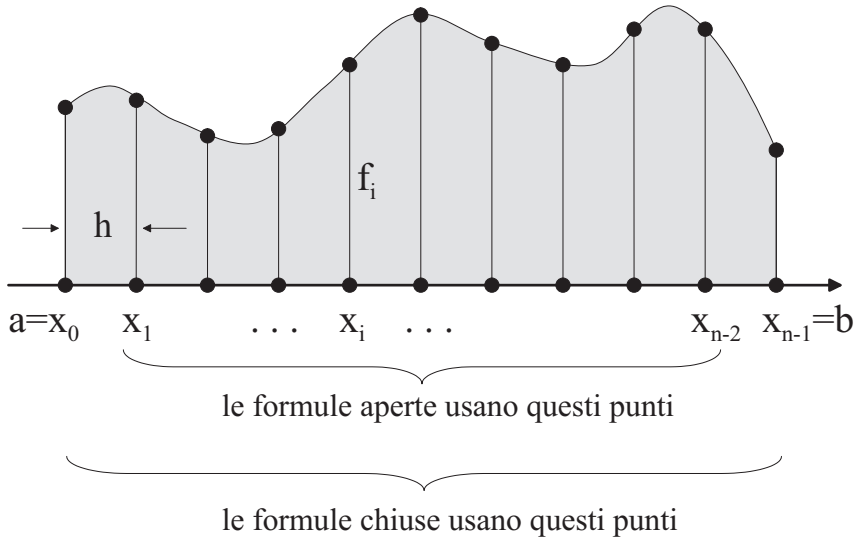


Figura 2.3: Discretizzazione della funzione integranda rispetto a intervalli aperti e chiusi

Le corrispondenti formule aperte si ottengono da quelle chiuse con un'opportuna correzione dei coefficienti (pesi) dei nodi agli estremi, secondo formule di estrapolazione. Senza entrare nel dettaglio (vedi, ad esempio, Press *et al.*, 1992) le corrispondenti formule aperte per il metodo dei trapezi e per il metodo di Simpson sono rispettivamente:

$$\int_{x_0}^{x_{n-1}} f(x)dx \approx h\left(\frac{3}{2}f_1 + f_2 + f_3 + \cdots + f_{n-3} + \frac{3}{2}f_{n-2}\right) \tag{2.9}$$

e

$$\begin{aligned}\int_{x_0}^{x_{n-1}} f(x)dx \approx h\Bigg(\frac{27}{12}f_1 + 0 \quad &+ \quad \frac{13}{12}f_3 + \frac{4}{3}f_4 + \frac{2}{3}f_5 + \ldots \\ \ldots \quad &+ \quad \frac{2}{3}f_{n-6} + \frac{4}{3}f_{n-5} + \frac{13}{12}f_{n-4} + 0 + \frac{27}{12}f_{n-2}\Bigg)\end{aligned} \tag{2.10}$$

Le formule semi-aperte sono ovvie combinazioni delle (2.9) e (2.10) con (2.6) e (2.8), rispettivamente.

2.2 Derivazione

2.2.1 Differenziali numerici

In parole molto povere si può dire che una funzione $f : (a, b) \to R$ è continua in un punto $x_0 \in (a, b)$ se, quando x si discosta di poco da x_0, $f(x)$ è poco distante da $f(x_0)$. Se si vuole valutare come varia la quantità

$$\frac{f(x) - f(x_0)}{x - x_0} \tag{2.11}$$

(coefficiente angolare della corda che passa per i punti $(x_0, f(x_0))$ e $(x, f(x))$ vicino al punto x_0 è naturale considerare il suo limite per $x \to x_0$. In particolare la quantità (2.11) è detta rapporto incrementale relativo alla funzione f nel punto x_0 e diciamo che f è derivabile in x_0 se

$$\lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \qquad \text{esiste finito.}$$

In tal caso chiamiamo il suo valore *derivata* di f in x_0 e scriviamo

$$f'(x_0) \qquad \text{oppure} \qquad \frac{df}{dx}(x_0)\ .$$

La derivata di f fornisce, vicino a x_0, una valutazione del modo in cui $f(x) - f(x_0)$ varia al variare di $x - x_0$. In altre parole, vicino a x_0, la quantità $f(x) - f(x_0)$ è approssimabile, in qualche senso, mediante la quantità $f'(x_0)(x - x_0)$ che è ovviamente di ben più facile trattazione. Tale quantità rappresenta geometricamente una retta tangente alla funzione

f nel punto x_0 (che rappresenta l'approssimazione del primo ordine della funzione in x_0). Allo stesso modo si possono definire le derivate di ordine superiore.

Il calcolo della derivata puntuale di una funzione può quindi essere approssimato dal suo rapporto incrementale. Per valutare l'errore che si commette con tale approssimazione vediamo prima l'errore su un singolo passo $h = (x - x_0)$ e poi estenderemo a l'errore sull'intero intervallo. Sia $f \in C^2$

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}h^2 f''(\xi) \qquad \xi \in [x_0, x_0 + h]$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{1}{2}hf''(\xi)$$

Poiché l'errore sul singolo passo h è una $O(h)$ e il numero di sottointervalli è $n = (b-a)/h$, l'errore complessivo sarà $O(1)$, ciò vuol dire che se $h \to 0$ l'errore risulta finito. Siamo pertanto in una situazione pessimistica in cui l'errore di troncamento sull'intero intervallo non viene ridotto riducendo il passo.

E' facile verificare che si giunge a conclusioni simili considerando il rapporto incrementale sinistro.

Supponiamo che $f \in C^3$ e scriviamo le approssimazioni di Taylor per $f(x_0+h)$ e $f(x_0-h)$:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + \frac{1}{6}f'''(\xi)h^3 \qquad \xi \in [x_0, x_0 + h]$$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{1}{2}f''(x_0)h^2 - \frac{1}{6}f'''(\eta)h^3 \qquad \eta \in [x_0 - h, x_0]$$

dalla differenza delle due espressioni si ottiene:

$$f(x_0 + h) - f(x_0 - h) = 2f'(x_0)h + \frac{1}{6}\left[f'''(\xi) + f'''(\eta)\right]h^3$$

Sapendo che per qualunque funzione continua in un intervallo la media di due suoi valori è uguale al valore della funzione in un punto intermedio.

$$f(x_0 + h) - f(x_0 - h) = 2f'(x_0)h + \frac{1}{3}f'''(\zeta)h^3 \qquad \zeta \in [x_0 - h, x_0 + h]$$

da cui si ottiene

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{1}{6}f'''(\zeta)h^2$$

che rappresenta il rapporto incrementale centrato.

Approssimando la derivata con questo rapporto incrementale l'errore sul singolo passo va con h^2 e l'errore complessivo, sull'intero intervallo, sarà $O(h)$ (tenderà a zero con h). L'errore che si commette approssimando la derivata con il rapporto incrementale centrato va a zero più rapidamente di quello associato ad un rapporto incrementale monolaterale (destro o sinistro), assicurando complessivamente un errore più piccolo. Diminuendo il

passo siamo sicuri che l'errore diminuisce, cosa che non era certa nel caso di differenze non centrate.

L'espressione approssimata per una derivata seconda si può ricavare con procedimento analogo sommando gli sviluppi di Taylor al quarto ordine della funzione f e ottenendo

$$f(x_0 + h) + f(x_0 - h) = 2f(x_0) + f''(x_0)h^2 + \frac{1}{24}\left[f^{iv}(\xi) + f^{iv}(\eta)\right] h^4$$

Questa espressione mi permette di calcolare una forma approssimata per la derivata seconda della funzione e di valutare l'errore ad essa associato:

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} - \frac{1}{12} f^{iv}(\zeta) h^2$$

Anche in questo caso l'errore che commetto va a zero con h^2, e l'errore globale andrà a zero con ordine h.

Un risultato analogo si ottiene ricavando l'espressione approssimata della derivata seconda come rapporto incrementale delle derivate prime:

$$f''(x_0) = \frac{f'(x_0)_+ - f'(x_0)_-}{h} = \frac{[f(x_0 + h) - f(x_0)] - [f(x_0) - f(x_0 - h)]}{h^2}$$

2.2.2 Derivazione numerica

L'approssimazione della derivata di una funzione in un punto mediante il suo rapporto incrementale suggerisce direttamente l'algoritmo:

1. scegliere un valore piccolo per il passo h
2. valutare $f(x + h)$
3. valutare $f(x)$ nel caso non fosse già disponibile tale valore
4. applicare la relazione

$$f'(x) \sim \frac{f(x + h) - f(x)}{h} \tag{2.12}$$

Un'applicazione indiscriminata di tale procedimento rischia tuttavia di portare a risultati non accurati. In particolare, un aspetto critico è rappresentato dalla scelta di h.

In generale l'approssimazione della derivata mediante differenziali numerici contiene due sorgenti di errore: (1) errori di troncamento e (2) errori di arrotondamento:

$$e = e_r + e_t$$

Come già evidenziato nel paragrafo precedente l'errore di troncamento deriva dai termini di ordine superiore nell'espressione in serie di Taylor:

$$f(x + h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \ldots$$

da cui

$$\frac{f(x+h)-f(x)}{h} = f' + \frac{1}{2}hf'' + \dots$$

per cui l'errore di troncamento nel caso di rapporto incrementale non centrato è dell'ordine di

$$e_t \sim |hf''(x)| \ .$$

L'errore di arrotondamento ha invece diversi contributi. Prima di tutto dobbiamo considerare l'errore di arrotondamento su h. Supponiamo, ad esempio, di avere il punto $x = 10.3$ e di scegliere "alla cieca" $h = 0.0001$. Né $x = 10.3$, né $x + h = 10.30001$ vengono rappresentati in modo esatto in forma binaria; ciascuno è pertanto rappresentato con un errore relativo ε_m caratteristico della rappresentazione floating point della macchina, il cui valore in aritmetica a singola precisione potrebbe essere dell'ordine di 10^{-7}. L'errore nell'effettivo valore di h, ovvero la differenza tra $(x+h)$ e x cosí come sono rappresentati nella macchina, è pertanto dell'ordine di $\varepsilon_m x$ e ciò comporta un errore relativo su h dell'ordine di

$$\sim \varepsilon_m \frac{x}{h} \sim 10^{-2} \ .$$

Pertanto, con queste ipotesi, applicando direttamente la (2.12) si avrebbe, nel migliore dei casi, almeno lo stesso errore relativo nella derivata.

Ecco dunque la lezione no.1: *Scegliere sempre h in modo che la differenza tra $(x + h)$ e x sia un numero rappresentabile esattamente nella macchina!*

Per fare ciò si può usare il seguente accorgimento:

$$\texttt{temp} = \texttt{x} + \texttt{h}$$

$$\texttt{h} = \texttt{temp} - \texttt{x}$$

Alcuni compilatori con ottimizzazione e alcuni calcolatori i cui processori floating point hanno una maggiore accuratezza interna di quella con cui i valori vengono memorizzati esternamente possono eludere questo accorgimento. In tal caso è generalmente sufficiente dichiarare `temp` come *volatile* o, altrimenti, fare una *call* di una funzione fittizia (`donothing`) tra le due relazioni:

$$\texttt{temp} = \texttt{x} + \texttt{h}$$

$$\texttt{donothing}(\texttt{temp})$$

$$\texttt{h} = \texttt{temp} - \texttt{x}$$

Queste operazioni forzano l'allocazione di `temp` nella memoria indirizzabile.

Quando h è rappresentato in modo esatto, l'errore di arrotondamento nel rapporto incrementale è

$$e_r \sim \varepsilon_f |f(x)/h|$$

dove ε_f è la precisione relativa (o errore relativo) sul calcolo della funzione f, (ovvero l'accuratezza con cui viene valutata la funzione f in un punto). Per una funzione "semplice" può essere approssimata con la precisione di macchina

$$\varepsilon_f \approx \varepsilon_m$$

ma per funzioni più complesse

$$\varepsilon_f > \varepsilon_m$$

In queste ipotesi e ricordando che l'errore totale $e = e_r + e_t$ è la somma dei due contributi e che entrambi i contributi dipendono dalla scelta di h, la scelta ottimale di h sarà quella che minimizza $e_r + e_t$. Infatti

$$e = e_r + e_t \sim \varepsilon_f \left| \frac{f(x)}{h} \right| + |hf''(x)|$$

ponendo

$$\frac{de}{dh} = 0$$

si ricava

$$h \sim \sqrt{\frac{\varepsilon_f f(x)}{f''(x)}} \approx \sqrt{\varepsilon_f} x_c \tag{2.13}$$

dove $x_c \equiv (f/f'')^{1/2}$ è la "curvature scale" della funzione f, o la "characteristic scale" su cui la funzione cambia. In assenza di informazioni si può sempre assumere $x_c = x$ (eccetto quando $x = 0$).

Scegliendo

$$h = \sqrt{\frac{\varepsilon_f f}{f''}}$$

l'accuratezza relativa nel calcolo numerico della derivata risulta

$$\frac{e_r + e_t}{|f'|} \sim \sqrt{\varepsilon_f} \left(\frac{ff''}{f'^2} \right)^{1/2} \sim \sqrt{\varepsilon_f} \tag{2.14}$$

assumendo che f, f', e f'' condividano tutte la stessa "characteristic length scale", il che corrisponde usualmente alla realtà. Dalla (2.14) si osserva che la relazione (2.12) permette di calcolare il valore approssimato della derivata con una precisione al più pari a $\sqrt{\varepsilon_f}$.

Come accennato nel paragrafo precedente una relazione più accurata per il calcolo numerico della derivata è il rapporto incrementale centrato

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{2.15}$$

In questo caso l'errore di troncamento è dell'ordine $e_t \sim h^2 |f'''|$. L'errore di arrotondamento e_r non cambia invece significativamente. Con un procedimento simile al caso precedente possiamo ricavare l'ottimale valore di h ricercando il minimo dell'errore totale $e = e_r + e_t$, da cui si ricava

$$h \sim \left(\frac{\varepsilon_f f}{f'''} \right)^{1/3} \sim (\varepsilon_f)^{1/3} x_c \tag{2.16}$$

In questo caso l'errore relativo sulla derivata risulta

$$\frac{(e_r + e_t)}{|f'|} \sim (\varepsilon_f)^{2/3} \frac{f^{2/3}(f''')^{1/3}}{f'} \sim (\varepsilon_f)^{2/3} \tag{2.17}$$

che tipicamente sarà un ordine di grandezza più piccolo di (2.14), nel caso di aritmetica a singola precisione, e due ordine di grandezza più piccolo nel caso di aritmetica a doppia precisione.

Dalla (2.13) e dalla (2.16) si può trarre la lezione no.2: *Scegliere h in modo che sia una potenza esatta di ε_f o di ε_m moltiplicata per una "characteristic scale" x_c.*

In ogni caso, l'errore ottenuto con i rapporti incrementali (2.12) e (2.15) rimane al di sopra della precisione di macchina ε_m o (nei casi peggiori) al di sopra dell'accuratezza ε_f con cui viene valutata la funzione.

Esistono naturalmente altri metodi che permettono di ottenere una precisione maggiore, ma la trattazione di questi metodi va al di là dello scopo di queste note e si rimanda a testi specialistici (vedi, ad esempio, Press *et al*, 1992). In generale questi metodi ricorrono al calcolo della funzione in più punti, nell'uso di polinomi interpolatori e risultano applicabili anche quando la f non è nota analiticamente e quando è presente un elevato rumore sul segnale.

Bibliografia

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

G. Monegato, *Fondamenti di calcolo numerico*, Libreria Editrice Universitaria Levrotto&Bella, Torino, 1990

O. Caligaris, P. Oliva, *Analisi Matematica I*, ECIG Genova (2^a ed.), 1986

B. Fadini, C. Savy, *Fondamenti di Informatica - fondamenti di programmazione*, Liguori editore, 1991

Capitolo 3

Ricerca delle radici di un'equazione

3.1 Introduzione

La risoluzione di generiche equazioni non lineari

$$f(x) = 0$$

costituisce un problema tipico del calcolo numerico, peraltro di difficile soluzione generale, giustificata dalla varietà di forme che tali equazioni possono assumere e dalla eventuale molteplicità delle radici.

Purtroppo, come vedremo nei prossimi paragrafi, la non linearità della $f(x)$ introduce particolari difficoltà, soprattutto in relazione al problema della convergenza dei metodi di risoluzione proposti, necessariamente di tipo iterativo.

Le radici di un'equazione non lineare $f(x) = 0$ non possono in generale venire espresse in "forma chiusa"; e anche quando ciò è possibile, la corrispondente espressione può risultare troppo complessa, o comunque non competitiva con altri modi di procedere. Pertanto, per risolvere numericamente equazioni non lineari siamo costretti a ricorrere a metodi approssimati. Questi ultimi sono necessariamente di tipo iterativo; partendo da una o più approssimazioni iniziali essi producono una successione $x_0, x_1, x_2, \ldots, x_n, \ldots$ convergente, quando le ipotesi richieste sono soddisfatte, alla radice incognita. Partendo da un valore iniziale di tentativo, un algoritmo migliorerà l'approssimazione della radice finché non è soddisfatto un criterio prefissato di convergenza. Per funzioni con variazioni "dolci" un buon algoritmo convergerà sempre alla soluzione purché sia stata fatta una scelta abbastanza "felice" del valore iniziale di tentativo. La scelta del valore iniziale deve essere guidata dal comportamento della funzione f. Lo sforzo di una scelta oculata sarà ricompensato con un minor costo computazionale dell'algoritmo e con una maggiore certezza della soluzione.

Nel seguito affrontiamo la costruzione di metodi per il calcolo di sole radici reali di una generica equazione non lineare.

3.2 Metodo di bisezione

Supponiamo che la funzione $f(x)$ sia continua nell'intervallo $[a_0, b_0]$ e che $f(a_0)f(b_0) < 0$. Queste ipotesi ci garantiscono l'esistenza di una radice nell'intervallo suddetto. Utilizzando tali informazioni ci proponiamo di costruire una successione di intervalli incapsulati $(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \supset \ldots$, tutti contenenti una radice dell'equazione $f(x) = 0$, con $(b_n - a_n) \to 0$ per $n \to \infty$ (v. Fig. 3.1).

Per esempio, sia $f(a_0) < 0$ e $f(b_0) > 0$. Gli intervalli successivi $(a_n, b_n), n = 1, 2, \ldots,$ vengono individuati con la strategia seguente. Dato (a_{n-1}, b_{n-1}), determiniamo il punto medio

$$m_n = \frac{1}{2}(a_{n-1} + b_{n-1}).$$

Supponiamo $f(m_n) \neq 0$, altrimenti m_n è radice. Esaminiamo il segno di $f(m_n)$ e poniamo

$$(a_n, b_n) = \begin{cases} (m_n, b_{n-1}) & \text{se } f(m_n) < 0 \\ (a_{n-1}, m_n) & \text{se } f(m_n) > 0 \end{cases}$$

Dopo n passi giungiamo all'intervallo (a_n, b_n), contenente la radice ξ cercata, di ampiezza

$$b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2} = \frac{b_{n-2} - a_{n-2}}{2^2} = \cdots = \frac{b_0 - a_0}{2^n}\ .$$

Come stima della radice ξ prendiamo

$$m_{n+1} = \frac{1}{2}(a_n + b_n)$$

così che

$$\xi = m_{n+1} + e_{n+1}, \qquad |\ e_{n+1}\ | < \frac{b_0 - a_0}{2^{n+1}}$$

La convergenza del metodo di bisezione è lenta; ad ogni passo "guadagnamo" una cifra binaria. Poiché $10^{-1} \simeq 2^{-3.3}$, in media ogni 3.3 passi guadagnamo una cifra decimale. Va però sottolineato che il metodo richiede solo la continuità della funzione $f(x)$ e la conoscenza del segno di $f(x)$ in $[a_0, b_0]$.

3.3 Criterio di convergenza

L'approssimazione della radice dell'equazione migliora ad ogni passo e occorre stabilire un criterio di convergenza che arresti il procedimento quando l'approssimazione raggiunge una precisione desiderata.

Ricordiamo tuttavia che i calcolatori rappresentano i numeri reali con un numero finito di cifre binarie. Pertanto anche se la funzione $f(x)$ passa analiticamente per lo zero, è possibile che quando viene valutata dal calcolatore, questa non assuma mai il valore zero per nessun argomento *floating point* rappresentabile dal calcolatore. Occorre dunque decidere quale accuratezza è possibile raggiungere sul valore della radice: una precisione assoluta di 10^{-6}

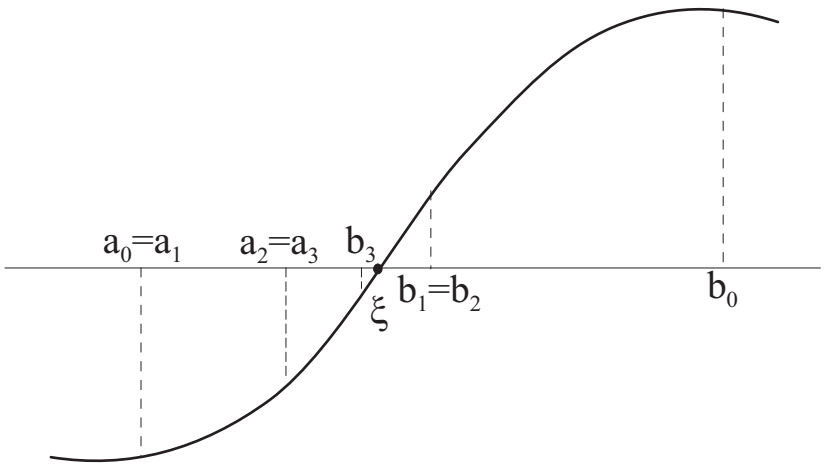


Figura 3.1: Rappresentazione grafica del metodo di bisezione

è ragionevole quando la radice cercata è nell'intorno di 1, ma è certamente irragiungibile se la radice è nell'intorno di 10^{26}. Si potrebbe pensare allora di definire un criterio di convergenza relativo (frazionario), ma questo non funzionerebbe per radici vicine allo zero.

In generale è opportuno specificare una tolleranza assoluta ε in modo che il procedimento iterativo si arresti quando l'intervallo diventa più piccolo di tale valore di tolleranza in termini assoluti. Di solito si sceglie

$$\varepsilon = \varepsilon_m \frac{|a_0| + |b_0|}{2}$$

dove ε_m è la precisione della macchina.

Esercizio proposto Scrivere l'algoritmo per determinare con tolleranza relativa fissata ε una radice dell'equazione $f(x) = 0$, contenuta nell'intervallo (a_0, b_0), mediante la tecnica di bisezione. Scrivere poi un programma C che realizzi tale algoritmo stampando a video i successivi intervalli incapsulati (a_n, b_n) e i relativi punti medi m_n generati dall'algoritmo nel caso

$$f(x) = \sinh(x) - \frac{1}{x} \qquad \text{e} \qquad (a_0, b_0) \equiv (0.5, 2).$$

3.4 Ordine di convergenza

Prima di proseguire nella costruzione di metodi alternativi più efficienti, introduciamo una misura della velocità di convergenza di una successione numerica.

Sia $x_0, x_1, x_2, \ldots, x_n, \ldots$ una successione convergente al valore ξ; poniamo $e_n = \xi - x_n$. Se esiste un numero $p \geq 1$ e una costante (reale) positiva c $(< \infty)$ tali che

$$\lim_{n\to\infty} \frac{\mid e_{n+1} \mid}{\mid e_n \mid^p} = c$$

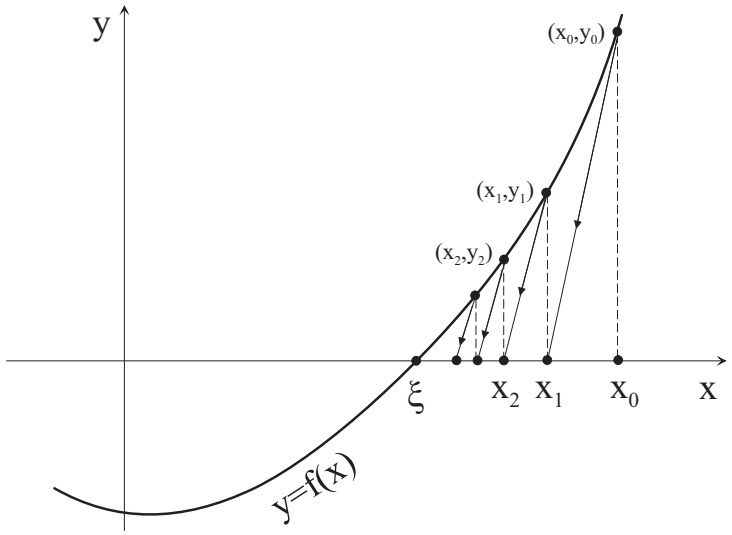


Figura 3.2: Rappresentazione grafica della generazione di una successione $x_0, x_1, \ldots, x_n \ldots$ convergente alla radice ξ

allora diciamo che la successione x_n ha ordine di convergenza p. Quando $p = 1, 2, 3$ la convergenza viene denominata lineare, quadratica, cubica. Nel caso della convergenza lineare è necessario che $c \leq 1$ (generalmente $c < 1$). La convergenza viene definita superlineare se $1 < p < 2$.

3.5 Metodi delle secanti, delle tangenti (Newton-Raphson) e altri

Dopo aver proposto nel paragrafo precedente un metodo con convergenza (quasi) lineare (bisezione), nelle pagine che seguono presentiamo la costruzione di metodi iterativi con ordine di convergenza superiore a 1.

Partendo da un'approssimazione iniziale x_0, graficamente possiamo pensare di generare i valori successivi $x_1, x_2, \ldots, x_n$ come in Fig. 3.2. Ossia, conduciamo dal punto iniziale (x_0, y_0), $y_0 = f(x_0)$, sulla curva $y = f(x)$ una retta con pendenza k_0, e prendiamo come nuova (e migliore) approssimazione x_1 l'intersezione di questa retta con l'asse x. Ripartiamo poi dal nuovo punto (x_1, y_1), $y_1 = f(x_1)$, con una seconda retta con pendenza k_1 e determiniamo l'intersezione x_2 di quest'ultima con l'asse x; e così via. In altri termini, ad ogni passo linearizziamo localmente il problema iniziale $f(x) = 0$, e come nuova approssimazione della radice ξ prendiamo la radice dell'equazione lineare

$$y_n + k_n(x - x_n) = 0, \qquad n = 0, 1, 2, \ldots,$$

cioè

$$x_{n+1} = x_n - \frac{f(x_n)}{k_n}, \qquad n = 0, 1, 2, \ldots.$$

Le direzioni $k_0, k_1, k_2, \ldots$ possono essere scelte in molti modi. Vediamo alcuni esempi:

1. Regula falsi:
$$k_n = \frac{y_n - y_0}{x_n - x_0}$$
2. Metodo delle secanti:
$$k_n = \frac{y_n - y_{n-1}}{x_n - x_{n-1}}$$
3. Metodo delle tangenti o di Newton-Raphson:
$$k_n = f'(x_n)$$

I primi due metodi richiedono la conoscenza di due approssimazioni iniziali, mentre nell'ultimo caso è indispensabile l'esistenza di $f'(x_n)$ per ogni n.

Esaminiamo i metodi proposti e poniamoci le seguenti domande: la successione $x_0, x_1, x_2, \ldots, x_n, \ldots$ converge alla radice ξ? Se si, qual è la velocità di convergenza? Ovvero quale è l'ordine del metodo?

Supponiamo che la radice ξ sia semplice, cioè $f'(\xi) \neq 0$, e approssimiamo la funzione con una retta sottesa tra due nodi $\{a, b\}$, non necessariamente distinti

$$f(x) = f(a) + (x - a)k + errore$$

dove $k = k(a, b)$ è il coefficiente angolare della retta. In particolare adottando la formula di interpolazione di Newton, possiamo scrivere

$$f(x) = f(a) + (x - a)k(a, b) + (x - a)(x - b)q(x, a, b)$$

In generale, se $f'(x)$ è definita in un intervallo contenente a, b e ξ si può dimostrare che

$$k(a, b) = f'(\alpha) \qquad \text{e} \qquad q(x, a, b) = 1/2 f''(\beta)$$

dove α e β sono due punti (non noti) del suddetto intervallo. Pertanto:

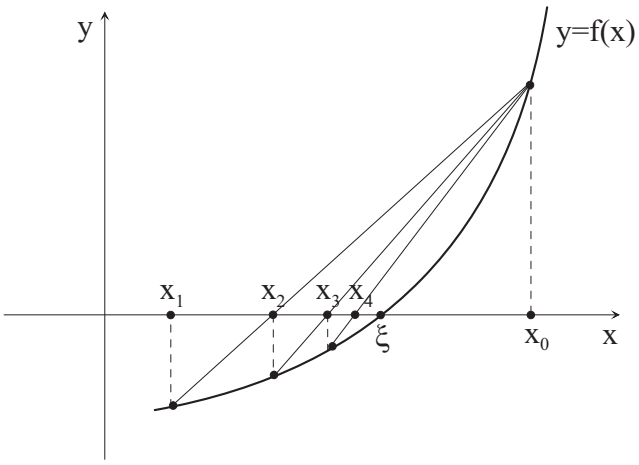
$$f(x) = f(a) + (x - a)f'(\alpha) + (x - a)(x - b)\frac{1}{2}f''(\beta)$$

Ponendo $f(\xi) = 0$

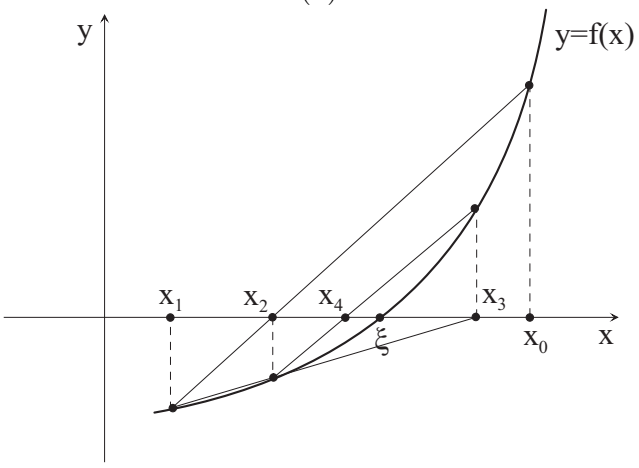
$$0 = f(a) + (\xi - a)f'(\alpha) + \frac{1}{2}(\xi - a)(\xi - b)f''(\beta)$$

dalla quale otteniamo

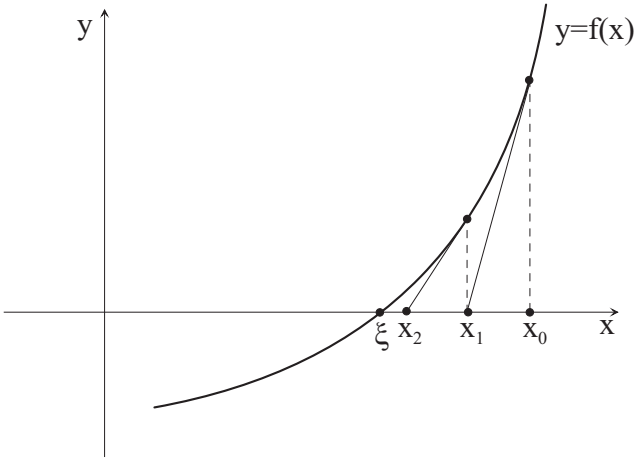
$$\xi = a - \frac{f(a)}{f'(\alpha)} - \frac{1}{2}(\xi - a)(\xi - b)\frac{f''(\beta)}{f'(\alpha)} \tag{3.1}$$



(a)



(b)



(c)

Figura 3.3: Rappresentazione grafica dei metodi della regula falsi (a), delle secanti (b) e delle tangenti (c)

Come approssimazione della radice ξ prendiamo l'ascissa

$$\hat{\xi} = a - \frac{f(a)}{f'(\alpha)} \tag{3.2}$$

a cui è associato l'errore

$$e = \xi - \hat{\xi} = -\frac{1}{2}(\xi - a)(\xi - b)\frac{f''(\beta)}{f'(\alpha)} \, .$$

Sia $f(\xi) = 0$ e $f'(\xi) \neq 0$. Supponendo inoltre che la funzione $f(x)$ sia di classe C^2 in un intorno della radice ξ, esiste allora un intorno (r, s) di ξ dove per ogni coppia $a, b \in (r, s)$

$$\frac{|f''(\beta)|}{2|f'(\alpha)|} \leq M < \infty$$

e possiamo pertanto scrivere

$$|e| \leq M|\xi - a||\xi - b| \, .$$

Scegliendo convenientemente in Eq.(3.1) i nodi a e b otteniamo dalla (3.2)[1] i metodi 1, 2 e 3. Infatti, ponendo $a = x_n$ e $b = x_0$ abbiamo la regula falsi, con $a = x_n$ e $b = x_{n-1}$ otteniamo il metodo delle secanti, mentre la scelta $a = b = x_n$ produce il metodo delle tangenti.

Definendo $e_n = \xi - x_n$, le corrispondenti espressioni per l'errore sono

$\|e_{n+1}\| \leq (M\|e_0\|)\|e_n\|$	per la regula falsi
$\|e_{n+1}\| \leq M\|e_{n-1}e_n\|$	per il metodo delle secanti
$\|e_{n+1}\| \leq M\|e_n\|^2$	per il metodo delle tangenti

Nelle ipotesi fatte la convergenza dei tre metodi è certamente assicurata se e_0 (ed e_1 nel caso del metodo delle secanti) è sufficientemente piccolo. Infatti, per la regula falsi

$$|e_{n+1}| \leq k|e_n| \leq k^{n+1}|e_0|, \qquad k = M|e_0|$$

e $\lim_{n\to\infty} e_{n+1} = 0$ quando $k < 1$; per il metodo delle secanti invece, se $|e_0|$ e $|e_1|$ sono entrambi minori di M^{-1}, ovvero

$$|e_0|M \leq k \qquad \text{e} \qquad |e_1|M \leq k \qquad\qquad \text{con} \qquad k < 1$$

[1] Dove ovviamente, $\hat{\xi} = x_{n+1}$

dall'espressione dell'errore otteniamo, per esempio,

$$\begin{aligned}
&|e_2| \le k|e_1| \\
&|e_3| \le k|e_2| \\
&|e_4| \le k^2|e_3| \\
&|e_5| \le k^3|e_4| \\
&|e_6| \le k^5|e_5| \\
&|e_{n+1}| \le k^{m_n}|e_n| \qquad \text{con} \qquad \lim_{n\to\infty} m_n = +\infty
\end{aligned}$$

donde

$$\lim_{n\to\infty} e_{n+1} = 0$$

La successione $\{m_n\}$ coincide con quella dei numeri di Fibonacci, che possiamo definire con la relazione $m_0 = 0, m_1 = 1, m_{i+2} = m_{i+1} + m_i$, $i = 0, 1, 2, \ldots$ Inoltre,

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = \lim_{n\to\infty} k^{m_n} = 0$$

per cui in questo caso la convergenza risulta almeno superlineare. Lasciamo al lettore l'esame della convergenza del metodo delle tangenti.

Determiniamo infine gli ordini di convergenza dei tre metodi.

Regula falsi

Poiché

$$e_{n+1} = -e_n e_0 \frac{f''(\beta)}{2f'(\alpha)} \qquad \text{con} \quad \xi < \alpha, \beta < x_0$$

abbiamo

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = |e_0| \left| \frac{f''(\beta)}{2f'(\alpha)} \right|$$

inoltre essendo $f'(\xi) \neq 0$, se α è sufficientemente vicino a ξ risulta $f'(\alpha) \neq 0$. Il metodo ha pertanto ordine di convergenza almeno $p = 1$. Quando $f''(\xi) \neq 0$ possiamo affermare che l'ordine è esattamente 1.

Secanti

In questo caso

$$e_{n+1} = -e_n e_{n-1} \frac{f''(\beta)}{2f'(\alpha)} \qquad \text{con} \quad x_{n-1} < \alpha, \beta < x_n$$

e

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}} = -\frac{f''(\xi)}{2f'(\xi)}$$

Tuttavia, per determinare l'ordine del metodo occorre individuare il numero reale $p \geq 1$ che permette di scrivere la relazione

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = c, \qquad 0 < c < \infty$$

È possibile dimostrare che

$$\left[\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n e_{n-1}|}\right]^{1/p} = \lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} \qquad p = \frac{\sqrt{5}+1}{2}$$

e quindi concludere che il metodo delle secanti ha ordine $p = 1.618\ldots$.

Tangenti o Newton-Raphson

Dalla relazione

$$e_{n+1} = -e_n^2 \frac{f''(x_n)}{2f'(x_n)}$$

deduciamo

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = -\frac{f''(\xi)}{2f'(\xi)}$$

L'ordine di questo metodo è pertanto $p = 2$ (assumendo $f''(\xi) \neq 0$, altrimenti $p > 2$). Osserviamo tuttavia che mentre i metodi della regula falsi e quello delle secanti richiedono ad ogni passo[2] la valutazione della $f(x)$ in un solo punto (l'ultimo trovato), il metodo di Newton per determinare x_{n+1} richiede la valutazione sia di $f(x_n)$ che di $f'(x_n)$.

Osservazioni

Il metodo delle secanti ha lo svantaggio che la radice non necessariamente rimane localizzata all'interno dell'intervallo. Per funzioni che non sono sufficientemente continue, non è pertanto garantita la convergenza dell'algoritmo. Addirittura in casi particolari questo diverge.

Il metodo di Newton-Raphson comporta la valutazione sia della $f(x)$ sia della derivata $f'(x)$. Il metodo può essere applicato anche usando una differenza numerica per approssimare il valore della derivata nel punto

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Tuttavia non è una procedura consigliata per le seguenti ragioni: (1) vengono effettuate due valutazioni della funzione per passo, quindi al più l'ordine superlineare di convergenza sarà solo di $\sqrt{2}$; (2)considerando un valore troppo piccolo per h dovremo fare i conti con gli errori di arrotondamento, se al contrario lo prendiamo troppo grande, l'ordine di convergenza risulterà solo lineare, come scegliendo la valutazione iniziale $f'(x_0)$ per tutti i passi successivi. Pertanto le prestazioni del metodo di Newton-Raphson con l'approssimazione numerica della derivata della funzione risultano inferiori a quello delle secanti.

[2]tranne in quello iniziale

Esercizio proposto Ripetere l'esercizio precedente utilizzando comparativamente i tre metodi iterativi presentati, prendendo $x_0 = 0.5$ e $x_1 = 1$ per i metodi della regula falsi e delle secanti e $x_0 = 0.5$ per quello delle tangenti.

3.6 Osservazioni sulla convergenza

Esaminiamo il significato del concetto di ordine di convergenza. Prima di tutto osserviamo che quando al passo n-esimo l'errore assoluto di x_n è $|e_n| \leq (1/2)10^{-k}$, cioè x_n ha k decimali corretti, abbiamo

$$|e_{n+1}| \simeq c \left(\frac{1}{2}10^{-k}\right)^p = \frac{c}{2^p}10^{-pk}$$

il numero di decimali corretti presenti in x_{n+1} è dunque dell'ordine di pk. La relazione precedente è in realtà una relazione limite; ciò significa che il numero di decimali corretti tende ad essere moltiplicato per p ad ogni passo solo per $n \to \infty$. Per valori finiti di n (e soprattutto nei primi passi) l'aumento di cifre dipende anche dalla costante moltiplicativa c_n presente nella relazione

$$|e_{n+1}| = c_n |e_n|^p \ , \qquad \lim_{n\to\infty} c_n = c \ .$$

La convergenza dei metodi delle secanti e delle tangenti è garantita quando, supposte soddisfatte le condizioni di regolarità sulla $f(x)$ richieste per la convergenza, l'approssimazione iniziale x_0 (e x_1 nel caso delle secanti) è "sufficientemente" vicina alla radice. Pertanto, tali metodi spesso si rivelano efficienti soprattutto per migliorare un'approssimazione "sufficientemente buona" ottenuta con un metodo di ordine 1 la cui convergenza è assicurata. Per esempio potremmo utilizzare il metodo di bisezione per determinare un'approssimazione con 1-2 cifre significative, e poi applicare il metodo di Newton-Raphson oppure quello delle secanti per ottenere con pochissime iterazioni la precisione desiderata.

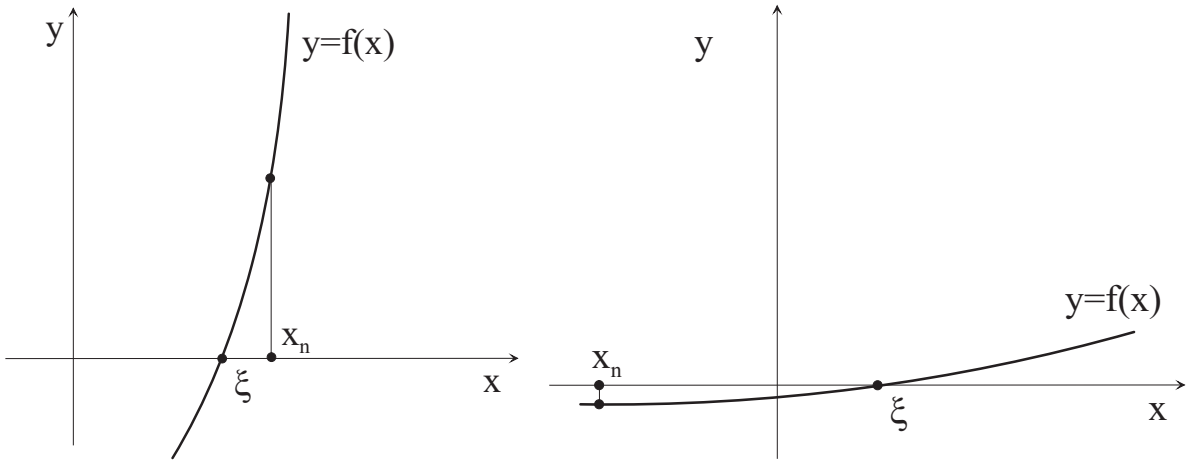
3.7 Test di convergenza

Quale criterio adottiamo per decidere se una data approssimazione x_n è sufficientemente accurata? Due sono i possibili criteri di arresto: il "controllo del residuo" ed il "controllo dell'incremento".

1. *Controllo del residuo:* il processo iterativo si arresta al minimo n per cui si abbia:

   $$|f(x_n)| \leq f_{\text{toll}}$$

   Si possono verificare situazioni in cui il test risulta o eccessivamente restrittivo o eccessivamente ottimistico (vedi, ad esempio, Fig. 3.4)



(a) $|e_n| \ll |f(x_n)|$ (b) $|e_n| \gg |f(x_n)|$

Figura 3.4: Due situazioni nelle quali il criterio basato sul residuo è troppo restrittivo o troppo ottimistico

2. *Controllo dell'incremento:* il processo iterativo si arresta non appena:

$$|x_n - x_{n-1}| \leq x_{\text{toll}} \qquad \text{oppure} \qquad |x_n - x_{n-1}| \leq x_{\text{toll}}|x_n|$$

Mentre questo test risulta valido per successioni con ordine $p > 1$, può invece provocare degli inconvenienti quando la convergenza è solamente lineare.

Quando non si ha la certezza della bontà di uno dei due test conviene includerli entrambi. Occorre inoltre precisare sempre un numero massimo di iterazioni e arrestare il processo di calcolo quando tale limite viene superato.

3.8 Determinazione di radici multiple

Le radici multiple, o radici molto vicine tra loro, costituiscono un vero problema, specialmente se la molteplicità è pari. In tal caso potrebbe non esserci un (evidente) cambiamento di segno della funzione con conseguenti difficoltà nella localizzazione della radice (ovvero nella determinazione dell'esistenza della radice). I metodi sinora presentati sono stati tutti esaminati supponendo che la radice incognita sia semplice, cioè $f'(\xi) \neq 0$. Che cosa succede quando la radice è *doppia*? O, più in generale, quando essa ha molteplicità $\mu > 1$, cioè quando $f(\xi) = f'(\xi) = \cdots = f^{(\mu-1)}(\xi) = 0$ e $f^{(\mu)}(\xi) \neq 0$? Esaminiamo il comportamento

del metodo di Newton-Raphson:

$$\begin{aligned} e_{n+1} &= e_n + \frac{f(\xi - e_n)}{f'(\xi - e_n)} = \\ &= e_n + \frac{f(\xi) - e_n f'(\xi) + \cdots + \frac{(-1)^{\mu-1}}{(\mu-1)!} e_n^{\mu-1} f^{(\mu-1)}(\xi) + \frac{(-1)^{\mu}}{\mu!} e_n^{\mu} f^{(\mu)}(\eta_n)}{f'(\xi) - e_n f''(\xi) + \cdots + \frac{(-1)^{\mu-2}}{(\mu-2)!} e_n^{\mu-2} f^{(\mu-1)}(\xi) + \frac{(-1)^{\mu-1}}{(\mu-1)!} e_n^{\mu-1} f^{(\mu)}(\delta_n)} = \\ &= e_n - \frac{\frac{e_n^{\mu}}{\mu!} f^{(\mu)}(\eta_n)}{\frac{e_n^{\mu-1}}{(\mu-1)!} f^{(\mu)}(\delta_n)} = e_n \left[1 - \frac{1}{\mu} \frac{f^{(\mu)}(\eta_n)}{f^{(\mu)}(\delta_n)} \right] , \end{aligned}$$

donde

$$\lim_{n \to \infty} \frac{e_{n+1}}{e_n} = \frac{\mu - 1}{\mu} \quad .$$

Pertanto, quando $\mu > 1$ il metodo risulta ancora convergente, ma l'ordine scende a $p = 1$. Per ristabilire la convergenza quadratica è sufficiente modificare la formula di Newton-Raphson come segue:

$$x_{n+1} = x_n - \mu \frac{f(x_n)}{f'(x_n)} \, . \tag{3.3}$$

La formula (3.3) è utilizzabile quando la molteplicità della radice è nota. Se invece la molteplicità non è nota possiamo considerare la nuova funzione $g(x) = f(x)/f'(x)$, per la quale $g(\xi) = 0$ e $g'(\xi) \neq 0$, e applicare ad essa il metodo delle tangenti:

$$x_{n+1} = x_n - \frac{f(x_n) f'(x_n)}{[f'(x_n)]^2 - f(x_n) f''(x_n)} \, .$$

La convergenza della successione $\{x_n\}$ è nuovamente quadratica; tuttavia, per poter applicare questa formula è necessario valutare anche $f''(x_n)$.

3.9 Metodi iterativi in generale

Supponiamo di riscrivere l'equazione in esame $f(x) = 0$ nella forma

$$x = g(x)$$

con $g(x)$ derivabile in un intorno della radice incapsulata ξ e tale che $\xi = g(\xi)$ se e solo se $f(\xi) = 0$.

Nota un'approssimazione iniziale x_0 della radice ξ possiamo utilizzare una formula iterativa del tipo

$$x_{n+1} = g(x_n), \qquad n = 0, 1, 2, \ldots \tag{3.4}$$

e generare una successione di valori $\{x_n\}$ a partire dal valore inixiale x_0. Se questa successione converge ad un punto $\bar{x}$, si ha per continuità

$$\bar{x} = g(\bar{x}) \qquad \text{ovvero} \qquad \bar{x} = \xi$$

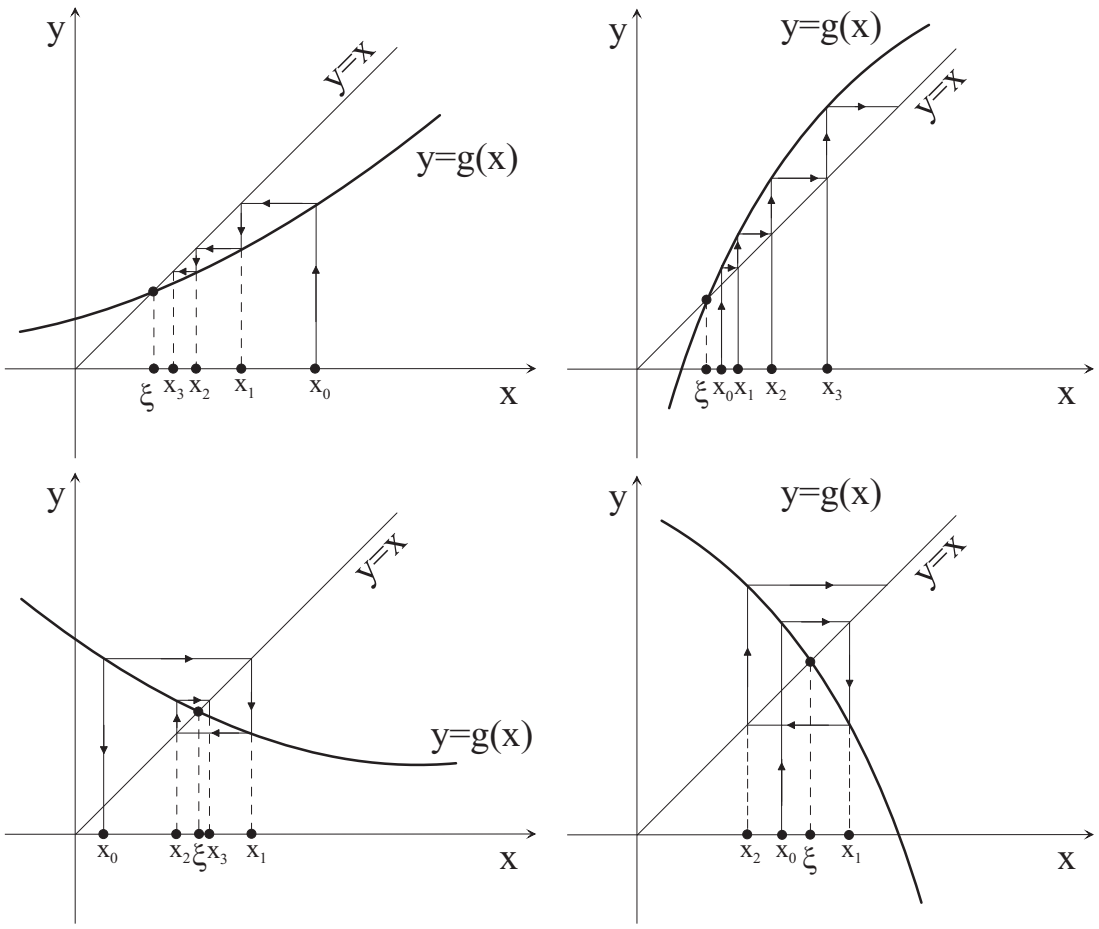


Figura 3.5: Rappresentazione grafica dell'applicazione di un metodo iterativo in quattro situazioni diverse

e quindi $\bar{x}$ è radice dell'equazione

$$x - g(x) = 0$$

oppure, ovviamente, di un'altra equazione che abbia la medesima radice ξ. Esaminiamo il significato geometrico della formula ricorsiva (3.4) in quattro situazioni diverse (v. Fig. 3.5). L'equazione $f(x) = 0$ può essere sempre riscritta nella forma $x = g(x)$; anzi ciò può essere fatto in un numero infinito di modi. Come mostrano i grafici in Fig. 3.5 la funzione $g(x)$, e in particolare il valore $g'(x)$, svolge un ruolo fondamentale; quest'ultimo è infatti il diretto responsabile della convergenza o meno della successione $\{x_n\}$. I criteri di scelta di $g(x)$ sono molteplici e danno luogo a formule iterative note con differenti denominazioni; la condizione di convergenza e la rapidità del metodo dipendono, oltre che da tale scelta, dal valore di tentativo e dalle caratteristiche della funzione nell'intorno della radice.

I metodi delle secanti e delle tangenti visti nei paragrafi precedenti possono essere interpretati come processi iterativi di tipo (3.4) con la scelta di

$$g(x) = x - \frac{1}{k_n} f(x) \ .$$

Bibliografia

G. Monegato, *Fondamenti di calcolo numerico*, Libreria Editrice Universitaria Levrotto&Bella, Torino, 1990

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

B. Fadini, C. Savy, *Fondamenti di Informatica - fondamenti di programmazione*, Liguori editore, 1991

A. Quarteroni, R. Sacco, F. Saleri, *Matematica Numerica*, Springer-Verlag Italia, 1998

O. Caligaris, P. Oliva, *Analisi Matematica I*, ECIG Genova (2^a ed.), 1986

Capitolo 4

Ricerca di massimi e minimi relativi di funzioni di una variabile

4.1 Introduzione

Molto spesso non è facile trovare i massimi e minimi relativi (od assoluti) di una funzione usando i metodi classici in quanto, ad esempio, non si possono facilmente valutare gli zeri della derivata prima o addirittura la funzione in esame non è espressa in forma analitica. In tali casi si ricorre a metodi per la determinazione numerica dei punti e dei valori di minimo. In generale, si può scegliere tra metodi che comportano la valutazione della sola funzione da minimizzare e metodi che richiedono anche la valutazione della sua derivata. Questi ultimi sono in qualche modo più potenti di quelli che usano solo i valori della funzione, ma non sempre così efficaci da compensare il costo addizionale del calcolo delle derivate.

Nel seguito si farà riferimento esclusivamente alla ricerca di minimi, osservando tuttavia che un problema di ricerca di massimo su $f(x)$ è equivalente ad un problema di ricerca di minimo su $-f(x)$.

4.2 Metodi iterativi

Possono essere considerati una trasposizione del metodo di bisezione applicato al problema della minimizzazione. Tali metodi si prefiggono di restringere, fino alla lunghezza desiderata, l'intervallo in cui è localizzato il minimo della funzione: in altre parole, assegnata una funzione $f(x)$ su un intervallo $[a, b]$, che ammetta ivi un minimo relativo interno, si cerca un intervallo più piccolo $[a_1, b_1] \subset [a, b]$ in modo che $f(x)$ ammetta ancora in $[a_1, b_1]$ un minimo relativo interno. Iterando il procedimento si può ridurre l'intervallo fino a trovare $[a_k, b_k]$ in modo che la sua ampiezza sia più piccola della precisione desiderata.

La possibilità di trovare l'intervallo $[a_1, b_1]$ a partire da $[a, b]$ è garantita da alcuni semplici risultati che illustriamo di seguito e dipende dalla scelta di opportuni punti in $[a, b]$.

4.2.1 Localizzazione di un minimo

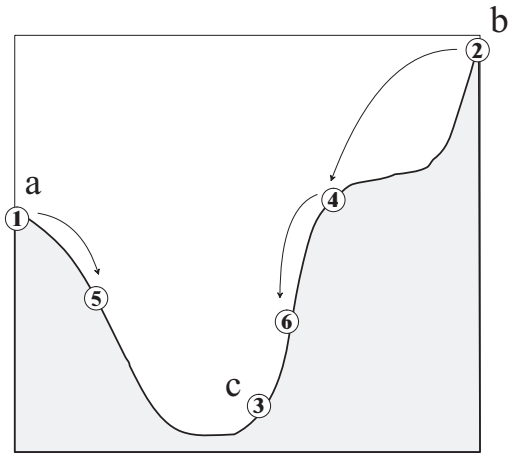


Figura 4.1: Successive localizzazioni del minimo. Il minimo è inizialmente localizzato dai punti 1,3,2. La funzione viene valutata in 4, che sostituisce 2; poi in 5, che sostituisce 1; poi in 6 che sostituisce 4. La regola impone che a ciascun passo si prenda un punto centrale che sia più basso dei due punti più "esterni". Al termine dei passi mostrati il minimo risulta localizzato dai punti 5,3,6

Se $f(x)$ è una funzione continua un punto di minimo viene localizzato quando esiste una terna di punti, $a < c < b$ (oppure $b < c < a$), tali che

$$f(c) < f(a) \qquad\qquad f(c) < f(b)$$

e con $f'(c) \neq 0$ (se $f'(c) = 0$ allora c è il punto di minimo cercato). In tal caso si può affermare che la funzione ha un minimo nell'intervallo (a, b). Il procedimento iterativo di ricerca del minimo comporta la scelta di un nuovo punto x compreso tra a e c o tra c e b. Per fissare le idee, supponiamo di fare quest'ultima scelta e valutiamo quindi $f(x)$. Se $f(c) < f(x)$ allora la nuova terna di punti che localizza il minimo è (a, c, x); altrimenti, se $f(c) > f(x)$, allora la nuova terna è (c, x, b). In ogni caso il punto medio della nuova terna è l'ascissa la cui ordinata è l'attuale miglior approssimazione del punto di minimo (v. Fig. 4.1).

Stabiliamo ora un criterio di scelta per passare dall'intervallo $[a, b] = [a_0, b_0]$ all'intervallo $[a_k, b_k]$ iterativamente.

Definizione Sia f continua e monomodale[1] in $[a, b]$, diciamo che è data una successione di intervalli $[a_k, b_k]$ localizzante un minimo relativo se:

- $[a_0, b_0] = [a, b]$;

[1] Una funzione $f(x)$ si dice monomodale su $I = [a, b]$ se esiste un numero $p \in I$ tale che $f(x)$ è decrescente in $[a, p]$ e crescente in $[p, b]$

- è assegnata una regola che permette di scegliere in (a_k, b_k) due punti, che indichiamo con α_k, β_k, (indichiamo questa regola con il nome di *regola di scelta*);

- è individuato $[a_{k+1}, b_{k+1}]$ nella seguente maniera

$$[a_{k+1}, b_{k+1}] = [a_k, \beta_k] \qquad \text{se} \quad f(\alpha_k) \leq f(\beta_k)$$
$$[a_{k+1}, b_{k+1}] = [\alpha_k, b_k] \qquad \text{se} \quad f(\alpha_k) > f(\beta_k)$$

Diciamo inoltre che la regola di scelta è *aurea* se accade che

$$\beta_{k+1} = \alpha_k \qquad \text{se} \quad f(\alpha_k) \leq f(\beta_k)$$
$$\alpha_{k+1} = \beta_k \qquad \text{se} \quad f(\alpha_k) > f(\beta_k)$$

Osserviamo che una regola di scelta aurea permette di introdurre ad ogni iterazione un solo nuovo punto e quindi causa ad ogni passo la necessità di un solo nuovo calcolo di funzione anziché due.

4.2.2 Sezione aurea

Considerato $[0, 1]$ l'intervallo iniziale, se $0.5 < r < 1$, allora $0 < 1 - r < 0.5$. In questo modo, l'intervallo viene diviso in tre sottointervalli $[0, 1 - r]$, $[1 - r, r]$, e $[r, 1]$. Usando un criterio di scelta è possibile "comprimere" l'intervallo iniziale da destra e ottenere il nuovo intervallo $[0, r]$ o da sinistra e ottenere il nuovo intervallo $[1-r, 1]$. Successivamente il nuovo intervallo viene suddiviso in altri tre sottointervalli con lo stesso criterio. Scegliamo r in modo che uno dei punti della nuova scelta corrisponda ad un punto della scelta precedente, come mostrato in Fig. 4.2. Ciò implica che

$$(1 - r) : r = r : 1$$

e pertanto r può essere determinato risolvendo l'equazione

$$1 - r = r^2$$

La soluzione r che soddisfa la condizione $0.5 < r < 1$ risulta

$$r = \frac{\sqrt{5} - 1}{2} \simeq 0.618034 \, .$$

Al fine di usare la regola di scelta aurea per la ricerca del minimo di $f(x)$ deve essere garantita l'esistenza di un valido punto di minimo della funzione nell'intervallo. Se $f(x)$ è monomodale in $[a, b]$, è possibile trovare un sottointervallo in cui $f(x)$ ha il suo valore di minimo. La regola di scelta aurea richiede che vengano usati due punti interni $\alpha_k = a_k + (1 - r)(b_k - a_k)$ e $\beta_k = a_k + r(b_k - a_k)$, dove r è la sezione aurea dell'intervallo. Tale suddivisione porta alla seguente scelta di punti $a_k < \alpha_k < \beta_k < b_k$. La condizione che $f(x)$ sia monomodale garantisce che i valori della funzione $f(\alpha_k)$ e $f(\beta_k)$ siano $< \max\{f(a_k), f(b_k)\}$. Possono verificarsi due casi (v. Fig. 4.3):

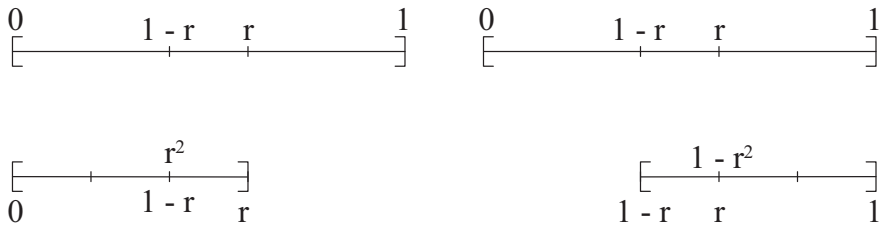


Figura 4.2: Criterio di scelta aurea dei sottointervalli

- se $f(\alpha_k) \leq f(\beta_k)$, il minimo sarà localizzato nel sottointervallo $[a_k, \beta_k]$, verrà sostituito b_k con β_k e proseguirà la ricerca nel vuovo sottointervallo;
- se $f(\alpha_k) > f(\beta_k)$, il minimo sarà localizzato in $[\alpha_k, b_k]$, si sostituisce a_k con α_k e si continua la ricerca.

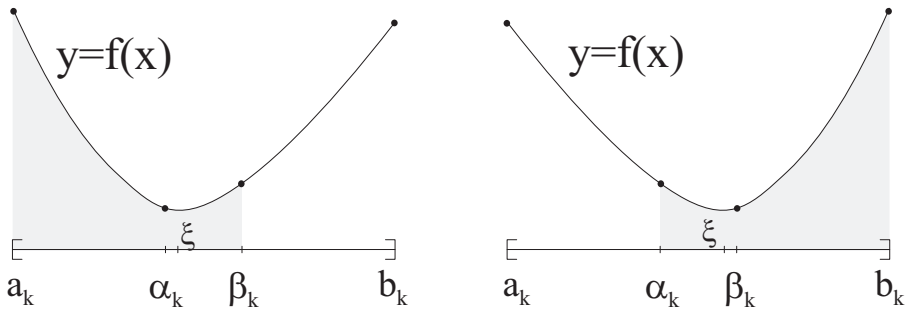


Figura 4.3: Processo di scelta dell'intervallo di localizzazione secondo la regola di scelta aurea

4.2.3 Criterio di arresto

Dopo n iterazioni la riduzione dell'intervallo di partenza è proporzionale ad un fattore

$$\left(\frac{\sqrt{5}-1}{2}\right)^n$$

infatti

$$b_n - a_n = r(b_{n-1} - a_{n-1}) = r^2(b_{n-2} - a_{n-2}) = \cdots = r^n(b_0 - a_0) \ .$$

Il processo iterativo può essere arrestato quando l'intervallo (a_n, b_n) risulta più piccolo di un valore prefissato di tolleranza[2] e si può considerare come approssimazione dell'ascissa

[2]Anche in questo caso, si può definire un criterio assoluto e uno relativo, rispettivamente:

$$|b_n - a_n| \leq \varepsilon \qquad \text{e} \qquad |b_n - a_n| \leq \varepsilon \frac{|b_n| + |a_n|}{2}$$

di minimo $\hat{\xi}_n$ il punto medio dell'intervallo (a_n, b_n)

$$\hat{\xi}_n = \frac{1}{2}(a_n + b_n)$$

o la sua sezione aurea

$$\hat{\xi}_n = a_n + r(b_n - a_n)$$

In ogni caso, l'errore che si commette

$$e_n = \xi - \hat{\xi}_n \leq r^{n+1}(b_0 - a_0) \ .$$

Si può inoltre osservare che il metodo ha una convergenza lineare (quindi piuttosto lenta) consentendo un guadagno medio di una cifra decimale circa ogni 4.78 passi.

4.3 Metodi basati sul calcolo della derivata

In linea di principio, il punto di minimo di una funzione può essere determinato cercando gli zeri della derivata prima (ad esempio, con le tecniche presentate nel Cap. 3) o del rapporto incrementale se l'espressione analitica della derivata prima non è disponibile. Come distinguere tuttavia tra "massimi" e "minimi"? L'ambiguità può essere rimossa valutando il segno della derivata seconda nel punto trovato, ma tale operazione aumenta il costo computazionale del metodo. E' quindi buona norma utilizzare sempre un metodo che permetta la localizzazione del minimo anche quando si ricorre all'informazione sulla derivata prima. L'unico modo per mantenere la localizzazione del minimo consiste nel valutare la funzione in tre punti e facendo in modo che il punto centrale sia sempre quello in cui la funzione assume il valore più basso rispetto agli altri due. L'informazione sulla derivata prima ci aiuterà a scegliere la nuova terna di localizzazione nel nuovo intervallo.

Nel prossimo paragrafo prenderemo in esame un metodo iterativo che, combinando l'approccio della localizzazione del minimo con le informazioni sulla derivata prima, converge al valore esatto in modo superlineare.

4.3.1 Metodo dell'interpolazione parabolica

Supponiamo che $f(x)$ sia monomodale in $[a, b]$ e che abbia un unico minimo in $x = \xi$. Assumiamo inoltre che $f'(x)$ sia definita in tutti i punti di (a, b). Sia $\xi_0 \in (a, b)$ il valore iniziale di approssimazione del punto di minimo. Si può osservare che (vedi Fig. 4.4):

- se $f'(\xi_0) < 0$ il punto di minimo ξ sarà alla destra di ξ_0, altrimenti
- se $f'(\xi_0) > 0$ il punto di minimo ξ sarà alla sinistra di ξ_0.

Nella pratica è consigliato utilizzare il criterio relativo.

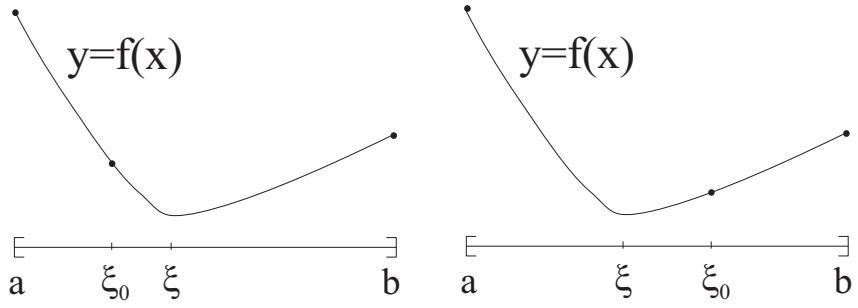


Figura 4.4: Uso della derivata prima nella ricerca del minimo di una funzione monomodale nell'intervallo $[a, b]$: se $f'(\xi_0) < 0$ il punto di minimo ξ sarà alla destra di ξ_0, altrimenti sarà alla sua sinistra

Localizzazione del minimo

Il primo obiettivo è quello di trovare una terna di punti

$$\xi_0, \qquad \xi_1 = \xi_0 + h \quad \text{e} \quad \xi_2 = \xi_0 + 2h$$

tali che

$$f(\xi_0) > f(\xi_1) \qquad e \qquad f(\xi_1) < f(\xi_2)\ . \tag{4.1}$$

Per fissare le idee, supponiamo che $f'(\xi_0) < 0$, allora

- $\xi_0 < \xi$ e
- il passo h deve essere scelto positivo ($h > 0$).

E' facile trovare un valore di h per cui sia soddisfatta la (5.14). Ad esempio, scelto $\xi_0 \in (a, b)$ si può procedere nel modo seguente:

1. Fissato il passo h si ottengono di conseguenza i punti

   $$\xi_1 = \xi_0 + h \quad e \quad \xi_2 = \xi_0 + 2h$$

   Si sceglie h in modo che ξ_1 e ξ_2 siano in (a, b).

2. Caso (a): se $f(\xi_0) > f(\xi_1)$ e $f(\xi_1) < f(\xi_2)$ allora abbiamo raggiunto lo scopo;

   Caso (b): se $f(\xi_0) > f(\xi_1)$ e $f(\xi_1) > f(\xi_2)$ allora $\xi_2 < \xi$. Occorre valutare la funzione in punti che sono più prossimi a b. Si pone $h = 2h$ e si ripete il procedimento;

   Caso (c): se $f(\xi_0) \leq f(\xi_1)$ allora abbiamo "saltato" il punto di minimo ξ poiché il passo h era troppo grande. Occorre dunque valutare la funzione in punti più prossimi a ξ_0. Si pone $h = h/2$ e si ripete il procedimento.

Se $f'(\xi_o) > 0$ il passo h deve essere scelto negativo ($h < 0$) e si può procedere in modo simile a prima.

Approssimazione parabolica

Alla fine del processo di localizzazione si avranno tre punti ξ_0, $\xi_0 + h$, $\xi_0 + 2h$ che soddisfano la (5.14). Per trovare il punto di minimo, assumendo che la funzione in un intorno di ξ sia abbastanza regolare, possiamo usare un'interpolazione quadratica e approssimare ξ con il punto di minimo della parabola ξ_{min} (vedi Fig. 4.5).

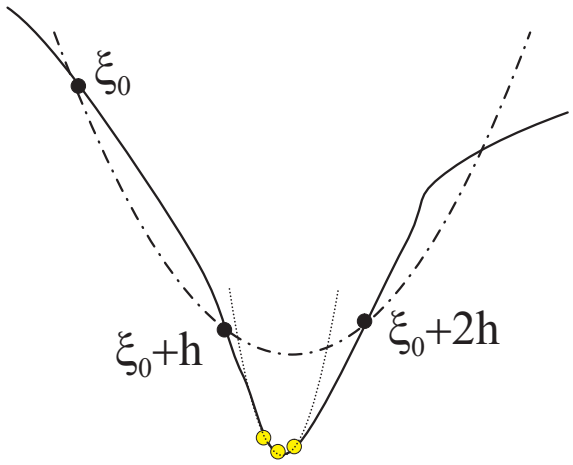


Figura 4.5: Rappresentazione grafica del metodo di approssimazione del minimo mediante interpolazione parabolica

Scrivendo il polinomio di Lagrange[3] per i tre punti che localizzano il minimo si ottiene

$$Q(x) = \frac{y_0(x-\xi_1)(x-\xi_2)}{2h^2} - \frac{y_1(x-\xi_0)(x-\xi_2)}{h^2} + \frac{y_2(x-\xi_0)(x-\xi_1)}{2h^2}$$

dove $y_0 = f(\xi_0), y_1 = f(\xi_1), y_2 = f(\xi_2)$. La derivata di $Q(x)$ è

$$Q'(x) = \frac{y_0(2x-\xi_1-\xi_2)}{2h^2} - \frac{y_1(2x-\xi_0-\xi_2)}{h^2} + \frac{y_2(2x-\xi_0-\xi_1)}{2h^2} \, .$$

[3]La formula di interpolazione di Lagrange permette di esprimere il valore di una funzione $y = f(x)$ attraverso un polinomio di grado $n-1$ che passa per i punti $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ della funzione:

$$y = \frac{y_1(x-x_2)\ldots(x-x_n)}{(x_1-x_2)\ldots(x_1-x_n)} + \frac{y_2(x-x_1)(x-x_3)\ldots(x-x_n)}{(x_2-x_1)(x_2-x_3)\ldots(x_2-x_n)} + \ldots$$

Risolvendo $Q'(x) = 0$ nella forma $Q'(\xi_0 + h_{min}) = 0$ si ottiene

$$\begin{aligned} 0 &= \frac{y_0[(2(\xi_0 + h_{min}) - \xi_1 - \xi_2]}{2h^2} \\ &- \frac{y_1[4(\xi_0 + h_{min}) - 2\xi_0 - 2\xi_2]}{2h^2} \\ &+ \frac{y_2[2(\xi_0 + h_{min}) - \xi_0 - \xi_1]}{2h^2} \, . \end{aligned} \tag{4.2}$$

Moltiplicando ciascun termine in (4.2) per $2h^2$ e raccogliendo a fattor comune h_{min} si può ricavare:

$$h_{min} = \frac{h(4y_1 - 3y_0 - y_2)}{4y_1 - 2y_0 - 2y_2} \, .$$

Il valore $\xi_{min} = \xi_0 + h_{min}$ rappresenta una migliore approssimazione di ξ rispetto a ξ_0. Possiamo quindi sostituire ξ_0 con ξ_{min}, scegliere un nuovo h (v. passo 1)[4] e ripetere il procedimento per determinare poi un nuovo h_{min} finché non è soddisfatto un criterio di arresto rispetto ad una tolleranza prefissata per l'approssimazione del punto di minimo.

Esercizi proposti

1. Trovare il minimo della funzione monomodale $f(x) = x^2 - \sin(x)$ in $[0, 1]$ utilizzando comparativamente il metodo iterativo basato sulla regola di scelta aurea e la valutazione numerica dello zero della derivata prima di $f(x)$ con uno dei metodi visti nel Cap. 3.

2. Trovare il minimo della funzione monomodale $f(x) = 8\cos^2 x + x^2 - 2x + 9$ in $[0, 3]$ utilizzando il metodo dell'interpolazione parabolica e considerando il punto iniziale $\xi_0 = 2.5$.

Bibliografia

O. Caligaris, P. Oliva, *Analisi Matematica I*, ECIG Genova (2^a ed.), 1986

J.H. Mathews, *Numerical methods*, Prentice Hall (2^a ed.), 1992

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

[4] Il segno di h dipenderà dal segno di $f'(\xi_{min})$ mentre il suo valore assoluto può essere scelto, ad esempio:

$$\begin{cases} |h| = |h_{min}| & \text{se } |h_{min}| < h \\ |h| = |h|/2 & \text{altrimenti} \end{cases}$$

Capitolo 5

Sistemi di equazioni lineari algebriche

5.1 Introduzione

Nel Cap. 2 abbiamo visto come determinare il valore di x che soddisfa una singola equazione $f(x) = 0$. Ora, invece, ci occuperemo della determinazione dei valori $x_1, x_2, \ldots, x_n$ che soddisfano simultaneamente un sistema di equazioni:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \ldots\ldots\ldots\ldots & \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

In generale, tali sistemi possono essere sia lineari che non lineari. In questo capitolo ci occuperemo delle equazioni lineari algebriche che hanno la forma generale:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{5.1}$$

ovvero

$$\sum_{j=0}^{n} a_{ij}x_j = b_i \qquad\qquad i = 0, \ldots, n$$

dove gli a_{ij} e i b_i sono coefficienti costanti e n è il numero delle equazioni.

5.1.1 Notazione matriciale

La notazione matriciale è ideale per rappresentare sistemi di equazioni lineari. Ad esempio la (5.1) può essere espressa come

$$A\mathbf{x} = \mathbf{b} \tag{5.2}$$

dove A è la matrice dei coefficienti di dimensione $n \times n$[1]:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

$\mathbf{b}$ è il vettore colonna $n \times 1$ dei termini noti (che indichiamo con il suo trasposto per comodità di lettura):

$$\mathbf{b}^T = [b_1\ b_2\ b_3 \dots b_n]$$

e $\mathbf{x}$ è il vettore colonna $n \times 1$ delle incognite:

$$\mathbf{x}^T = [x_1\ x_2\ x_3 \dots x_n]$$

In queste ipotesi, l'esistenza e l'unicità della soluzione di (5.1) è garantita se vale una delle seguenti condizioni (equivalenti):

1. A è invertibile;
2. rango$(A) = n$;
3. il sistema omogeneo $A\mathbf{x} = \mathbf{0}$ ammette come unica soluzione quella nulla.

La soluzione formale si ottiene per mezzo dell'algebra delle matrici, moltiplicando (a sinistra) entrambi i membri dell'equazione per l'inversa di A:

$$A^{-1}A\mathbf{x} = A^{-1}\mathbf{b}$$

Data che $A^{-1}A$ è uguale alla matrice unità, l'equazione diventa

$$\mathbf{x} = A^{-1}\mathbf{b}\ .$$

L'equazione, quindi, è stata risolta rispetto a $\mathbf{x}$. Talvolta può essere utile aggiungere $\mathbf{b}$ ad A. Per esempio, se $n = 3$, questa operazione dà come risultato una matrice 3×4:

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right] \tag{5.3}$$

Questa rappresentazione dei sistemi di equazioni è utile in quanto diverse tecniche per la risoluzione dei sistemi prevedono l'esecuzione di identiche operazioni su ogni riga di coefficienti più il corrispondente termine noto. Così come è costruita nella (5.3), diventa possibile manipolare una sola riga della matrice "aggiunta" anziché ripetere le operazioni su ogni riga della matrice dei coefficienti e sul corrispondente elemento del vettore dei termini noti.

[1]Nel seguito considereremo solo sistemi in cui il numero delle incognite coincide con quello delle equazioni, ovvero matrici A quadrate

5.1.2 Equazioni lineari algebriche nell'ingegneria

Molte delle equazioni fondamentali utilizzate nella progettazione ingegneristica sono derivate da leggi di conservazione (v. Tab. 5.1). Alcune grandezze che compaiono comunemente in tali leggi sono la massa, la forza, l'energia e la quantità di moto. In termini matematici, questi principi portano a equazioni di continuità o di bilancio che pongono in relazione il comportamento del sistema, rappresentato dai livelli o dai cambiamenti delle quantità che compaiono nel modello, con le proprietà o caratteristiche del sistema stesso e gli stimoli esterni agenti su di esso.

Principio fondamentale	Variabile dipendente	Variabile indipendente	Parametri
Conservazione del calore	Temperatura	Tempo e posizione	Proprietà termiche del materiale, geometria del sistema
Conservazione della massa	Concentrazione o quantità di massa	Tempo e posizione	Comportamento chimico del materiale, coefficienti di trasferimento di massa, geometria del sistema
Bilancio di forze	Intensità e direzione delle forze per ottenere l'equilibrio	Tempo e posizione	Resistenza del materiale, proprietà strutturali e configurazione del sistema
Conservazione dell'energia	Variazioni dell'energia cinetica e potenziale del sistema	Tempo e posizione	Proprietà termiche, massa e geometria del sistema
Leggi del moto di Newton	Accelerazione, velocità e posizione	Tempo e posizione	Massa e geometria del sistema, parametri dissipativi come il coefficiente di attrito e quello di resistenza aereodinamica
Leggi di Kirchhoff	Correnti e tensioni in un circuito elettrico	Tempo	Proprietà elettriche del sistema come resistenza, capacità e induttanza

Tabella 5.1: Principi fondamentali applicabili ai problemi di progettazione in ingegneria

In generale, quando un sistema è composto di più componenti, il suo comportamento

risulta descritto da un sistema di equazioni interdipendenti che devono essere risolte simultaneamente. Le varie equazioni dipendono tra di loro in quanto le singole parti che compongono il sistema si influenzano a vicenda. In Fig. 5.1, per esempio, il blocco 4 riceve ingressi dai blocchi 2 e 3: il suo comportamento, quindi, dipende anche dallo stato di questi altri due blocchi.

Quando queste dipendenze vengono espresse matematicamente, le equazioni risultanti assumono spesso la forma algebrica lineare (cfr. (5.1)), dove le $\mathbf{x}$ in genere corrispondono all'ampiezza e alla velocità di variazione dei parametri che descrivono il sistema. Le a_{ij} rappresentano tipicamente le proprietà e le caratteristiche che danno luogo alle interazioni tra i componenti del sistema. Le $\mathbf{b}$, infine, di solito rappresentano gli stimoli esterni agenti sul sistema.

Le equazioni differenziali derivate dalle leggi di conservazione descrivono l'andamento delle variabili dipendenti per sistemi di questo tipo. Tali equazioni differenziali possono essere integrate numericamente convertendole in un equivalente sistema di equazioni algebriche. La soluzione di questi sistemi rappresenta una delle principali aree di applicazione ingegneristica dei metodi descritti nei prossimi paragrafi. Queste equazioni sono interdipendenti in quanto il valore delle variabili in un punto del sistema dipende dai valori delle variabili nelle "regioni" contigue.

Oltre ai sistemi fisici, i sistemi di equazioni lineari algebriche trovano applicazione in una varietà di problemi matematici nei quali una funzione deve soddisfare diverse condizioni contemporaneamente. Ogni condizione dà luogo a un'equazione che contiene coefficienti noti e incognite. Le tecniche discusse in questo capitolo permettono di determinare il valore delle incognite quando le equazioni sono algebriche e lineari.

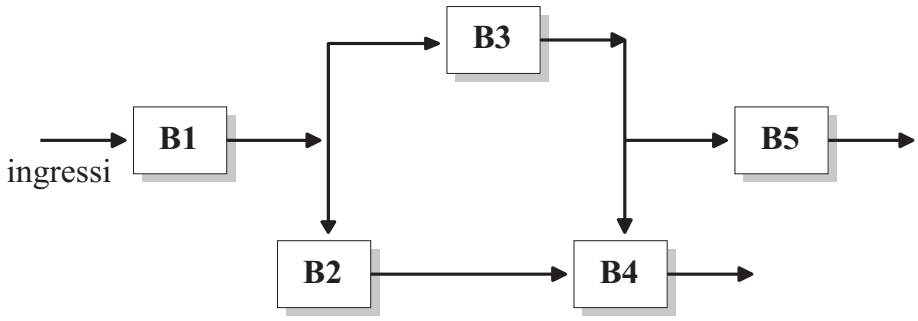


Figura 5.1: Rappresentazione schematica di un sistema a più componenti

Osservazione Scrivendo una singola equazione di bilancio per ogni parte del sistema fisico, si arriva a un insieme di equazioni che definisce l'andamento dei vari parametri descrittivi dell'intero sistema. Queste equazioni sono interdipendenti, nel senso che ogni equazione contiene una o più variabili che compaiono anche nelle altre equazioni. In molti casi, questi sistemi sono lineari e possono essere descritti nella forma

$$A\mathbf{x} = \mathbf{b} \, . \tag{5.4}$$

Nelle equazioni di bilancio, i termini della (5.4) hanno un'interpretazione fisica ben precisa: $\mathbf{x}$ rappresenta lo stato o la risposta del sistema, cioè quanto interessa determinare; il vettore dei termini noti $\mathbf{b}$ contiene quei dati del bilancio che sono indipendenti dal comportamento del sistema e che, pertanto, sono costanti (in molti casi, quindi, gli elementi di $\mathbf{b}$ rappresentano le forze esterne o altri stimoli che perturbano il sistema); la matrice dei coefficienti A, infine, contiene solitamente i parametri che descrivono l'interazione o l'accoppiamento esistente tra le parti del sistema fisico. Per quanto detto, la (5.4) può essere espressa anche nella forma

$$[Interazioni][Risposte] = [Stimoli]$$

Usando la matrice inversa si ottiene un risultato particolarmente interessante:

$$\mathbf{x} = A^{-1}\mathbf{b}$$

ovvero

$$\begin{aligned} x_1 &= a'_{11}b_1 + a'_{12}b_2 + \cdots + a'_{1n}b_n \\ x_2 &= a'_{21}b_1 + a'_{22}b_2 + \cdots + a'_{2n}b_n \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n &= a'_{n1}b_1 + a'_{n2}b_2 + \cdots + a'_{nn}b_n \end{aligned}$$

dove a'_{ij} sono gli elementi di A^{-1}. Troviamo, così, che la matrice inversa, oltre a permettere il calcolo della soluzione, è di per sè stessa dotata di utili proprietà: ciascuno dei suoi elementi, infatti, rappresenta la risposta di una singola parte del sistema a uno stimolo unitario proveniente da una delle parti che compongono il sistema, in assenza di altri stimoli. Osserviamo che le leggi così espresse sono linerari e, quindi, valgono i principi di sovrapposizione degli effetti e di proporzionalità: il *principio di sovrapposizione degli effetti* stabilisce che, se un sistema è sottoposto a un certo numero di stimoli differenti (le $\mathbf{b}$), è possibile calcolare la risposta a ogni stimolo separatamente e determinare poi la risposta totale come somma delle risposte individuali; il *principio di proporzionalità*, invece, stabilisce che, moltiplicando uno stimolo per una costante, si ottiene una risposta pari alla risposta originale moltiplicata per la stessa costante. Pertanto, il coefficiente a'_{11} è una costante di proporzionalità che dà il valore di x_1 dovuto a un valore unitario di b_1. Questo risultato è indipendente dagli effetti di b_2 e b_3 su x_1; di questi effetti, infatti, si tiene conto per mezzo dei coefficienti a'_{1j}, $j = 2, \ldots, n$. Pertanto, possiamo trarre la conclusione generale che gli elementi a'_{ij} della matrice inversa rappresentano il valore di x_i dovuto a un b_j unitario. La matrice inversa, quindi, è un potente strumento per la comprensione delle interazioni stimolo-risposta tra le parti che compongono un sistema fisico che spesso risultano non intuitive anche nel caso di sistemi semplici.

5.1.3 Interpretazione geometrica

La soluzione di un sistema di equazioni lineari in n incognite può essere interpretata come l'intersezione di iperpiani in uno spazio a n dimensioni. Per sistemi di piccole dimensioni l'interpretazione geometrica fornisce anche un metodo grafico per la risoluzione del sistema.

Nel caso di due equazioni è possibile ottenere una soluzione tracciando il grafico delle equazioni stesse in coordinate cartesiane con un asse corrispondente a x_1 e l'altro a x_2. Dato che ci occupiamo di sistemi lineari, ogni equazione rappresenta una retta. Ciò può essere facilmente dimostrato nel caso generale dato dalle equazioni:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned}$$

Risolviamo entrambe le equazioni rispetto a x_2:

$$x_2 = -\frac{a_{11}}{a_{12}}x_1 + \frac{b_1}{a_{12}}$$

$$x_2 = -\frac{a_{21}}{a_{22}}x_1 + \frac{b_2}{a_{22}}$$

Come si può vedere, le equazioni hanno ora la forma canonica per la rappresentazione delle rette: $x_2 = (\text{pendenza})x_1 + \text{intercetta}$. Queste rette possono essere riportate graficamente in coordinate cartesiane con x_2 come ordinata e x_1 come ascissa: i valori di x_1 e x_2 all'intersezione delle due rette rappresentano la soluzione.

Nel caso di un sistema di tre equazioni in tre incognite, a ogni equazione corrisponde un piano in un sistema cartesiano ortogonale tridimensionale; il punto d'intersezione comune a tutti e tre i piani rappresenta la soluzione.

5.1.4 Regola di Cramer

La regola di Cramer stabilisce che il valore di ciascuna incognita di un sistema di equazioni lineari algebriche può essere ottenuto tramite la seguente formula:

$$x_j = \frac{\Delta_j}{\det(A)}\,, \qquad\qquad j = 1, \ldots, n$$

dove $\det(A)$ è il determinante della matrice dei coefficienti A e Δ_j è il determinante della matrice ottenuta sostituendo la j-esima colonna di A con il termine noto $\mathbf{b}$. Applicando, ad esempio, la formula ad un sistema di tre equazioni e tre incognite

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

si ottiene

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{\det(A)} \qquad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}}{\det(A)} \qquad x_3 = \frac{\begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}}{\det(A)}$$

Se le equazioni sono più di tre, l'uso della regola di Cramer non è più conveniente, poiché la quantità di calcoli necessari cresce molto velocemente con la dimensione del problema. E' pertanto necessario cercare delle alternative più efficienti per l'implementazione su calcolatore. Queste tecniche saranno l'oggetto dei prossimi paragrafi.

5.1.5 Metodi per la soluzione al calcolatore

La formula di Cramer è di scarso interesse numerico. Se infatti si suppone di calcolare i determinanti tramite una formula ricorsiva[2], il costo computazionale richiesto dalla regola di Cramer è dell'ordine di $(n+1)!$ operazioni floating point (*flops*) ed è di conseguenza inaccettabile anche per piccole dimensioni di A (ad esempio, un calcolatore in grado di effettuare 10^9 *flops* al secondo, impiegherebbe $9.6 \cdot 10^{47}$ anni per risolvere un sistema lineare di sole 49 equazioni).

Per questo motivo sono stati sviluppati metodi numerici alternativi alla regola di Cramer, che vengono detti *diretti* se conducono alla soluzione del sistema dopo un numero finito di passi o *iterativi* se ne richiedono (teoricamente) un numero infinito. Per i primi, il numero di operazioni da eseguire è tipico dell'algoritmo di calcolo, per i secondi risulta dipendente *anche* dalla precisione richiesta, dai valori iniziali di tentativo e dalle caratteristiche della matrice dei coefficienti. Anticipiamo fin d'ora che la scelta tra un metodo diretto ed un metodo iterativo non dipenderà soltanto dall'efficienza teorica dello schema, ma anche dal particolare tipo di matrice, dall'occupazione di memoria richiesta ed, in ultima analisi, dall'architettura del calcolatore utilizzato.

Nei prossimi paragrafi prenderemo in esame, fra i metodi diretti

- il metodo dell'eliminazione gaussiana;
- il metodo di Gauss-Jordan;
- la decomposizione LU;

e, fra quelli iterativi

- il metodo di Jacobi;
- il metodo di Gauss-Seidel.

[2]Denotando con A_{ij} la matrice di ordine $n-1$ ottenuta da A per la soppressione della i-esima riga e della j-esima colonna, diciamo *minore complementare* associato all'elemento a_{ij} il determinante della matrice A_{ij}. Se indichiamo con $\Delta_{ij} = (-1)^{i+j} \det(A_{ij})$ il *cofattore* dell'elemento a_{ij}, per il calcolo effettivo dei determinanti si può usare la seguente relazione ricorsiva

$$\det(A) = \begin{cases} a_{11} & \text{se } n = 1, \\ \sum_{j=1}^{n} \Delta_{ij} a_{ij} \ , & \text{per } n > 1, \end{cases}$$

detta *formula di sviluppo del determinante secondo la riga i-esima* o *regola di Laplace*. Se A è una matrice quadrata di ordine n invertibile allora

$$A^{-1} = \frac{1}{\det(A)} C$$

essendo C la matrice di elementi Δ_{ji}, $i = 1, \ldots, n$; $j = 1, \ldots, n$.

5.2 Eliminazione gaussiana

5.2.1 Premessa

Nonostante sia una delle prime tecniche per la risoluzione di sistemi di equazioni lineari algebriche, l'eliminazione gaussiana è ancora uno degli algoritmi più efficienti per lo studio di molti problemi d'ingegneria. Oltre alla sua utilità nelle applicazioni, questo metodo consente di affrontare aspetti più generali, come gli errori di arrotondamento, la normalizzazione e il condizionamento dei sistemi. Inoltre questo algoritmo rappresenta la forma generale di qualsiasi metodo diretto per la risoluzione dei sistemi lineari, tutti basati sulla *tecnica dell'eliminazione delle incognite*.

L'approccio algebrico dell'eliminazione delle incognite può essere illustrato prendendo come esempio il caso di due equazioni in due incognite:

$$a_{11}x_1 + a_{12}x_2 = b_1 \tag{5.5}$$
$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{5.6}$$

Le equazioni vengono moltiplicate per un'opportuna costante in modo che una delle due incognite possa essere eliminata quando le equazioni vengono combinate. A questo punto ciò che rimane è una sola equazione in un'incognita che può essere risolta direttamente; il valore trovato viene poi sostituito in una delle equazioni originali per determinare l'incognita che era stata eliminata.

Per esempio, moltiplicando la (5.5) per a_{21} e la (5.6) per a_{11}:

$$a_{11}a_{21}x_1 + a_{12}a_{21}x_2 = b_1a_{21} \tag{5.7}$$
$$a_{21}a_{11}x_1 + a_{22}a_{11}x_2 = b_2a_{11} \tag{5.8}$$

Sottraendo la (5.7) dalla (5.8) eliminiamo x_1 dalle due equazioni e otteniamo

$$a_{22}a_{11}x_2 - a_{12}a_{21}x_2 = b_2a_{11} - b_1a_{21}$$

che può essere risolta rispetto a x_2:

$$x_2 = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{12}a_{21}} \tag{5.9}$$

La (5.9), sostituita nella (5.5), ci permette di calcolare x_1:

$$x_1 = \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - a_{12}a_{21}} \tag{5.10}$$

L'applicazione dell'eliminazione delle incognite può essere estesa a sistemi con più di due o tre equazioni. Il gran numero di calcoli necessario nel caso di grandi sistemi rende molto scomodo il metodo se la risoluzione deve essere ottenuta manualmente. Come vedremo, però, questa tecnica si presta ad essere espressa in modo sistematico: è così possibile realizzare programmi che implementino efficientemente il metodo su calcolatore.

5.2.2 Formulazione semplificata

La procedura usata dalla tecnica dell'eliminazione delle incognite consiste essenzialmente di due passi:

1. Le equazioni sono manipolate in modo da eliminare una delle incognite. Lo scopo di questo passo di *eliminazione* è quello di ottenere, alla fine del procedimento, un'equazione in una sola incognita.

2. A questo punto, si risolve l'equazione rimasta e il suo risultato viene *sostituito all'indietro* in una delle equazioni originali per ottenere l'incognita eliminata.

Questo semplice approccio può essere esteso a sistemi di grandi dimensioni: è sufficiente sviluppare una tecnica sistematica di eliminazione delle incognite e di sostituzione all'indietro dei valori trovati. La più comune di queste tecniche è l'*eliminazione gaussiana* o *metodo di Gauss*. Anche se queste tecniche possono essere implementate senza difficoltà su un personal computer, è necessario prevedere alcune modifiche per migliorare l'affidabilità dell'algoritmo; in particolare, bisogna evitare che il calcolatore tenti di eseguire una divisione per zero. Per semplicità di esposizione, la formulazione semplificata dell'algoritmo di eliminazione gaussiana descritto qui di seguito non prende in considerazione questa eventualità. Le ulteriori caratteristiche necessarie per realizzare un programma di effettiva utilità verranno descritte nei paragrafi successivi.

Vogliamo costruire un algoritmo generale per la risoluzione di un qualunque sistema di n equazioni in n incognite:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 && (5.11)\\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 && (5.12)\\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & && \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n && (5.13) \end{aligned}$$

Come nel caso di due equazioni, la tecnica da applicare ai sistemi di n equazioni prevede due fasi: l'eliminazione delle incognite e la loro determinazione per mezzo della sostituzione all'indietro.

Eliminazione in avanti delle incognite

In questa prima fase vogliamo trasformare l'insieme di equazioni in un sistema con matrice dei coefficienti triangolare superiore. Il primo passo della procedura consiste nel dividere la prima equazione (5.11) per il coefficiente a_{11} della prima incognita:

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n = \frac{b_1}{a_{11}}$$

Questa operazione viene detta normalizzazione e, in questo caso, il suo scopo è di rendere uguale a 1 il primo coefficiente della prima equazione. Successivamente, si moltiplica

l'equazione così normalizzata per a_{21}, il primo coefficiente della seconda equazione (5.12):

$$a_{21}x_1 + \left(a_{21}\frac{a_{12}}{a_{11}}\right)x_2 + \cdots + \left(a_{21}\frac{a_{1n}}{a_{11}}\right)x_n = a_{21}\frac{b_1}{a_{11}} \tag{5.14}$$

Osserviamo che il primo termine della prima equazione è ora identico al primo termine della seconda equazione; pertanto, possiamo eliminare la prima incognita dalla seconda equazione sottraendo la (5.14) dalla (5.12):

$$\left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)x_2 + \cdots + \left(a_{2n} - a_{21}\frac{a_{1n}}{a_{11}}\right)x_n = b_2 - a_{21}\frac{b_1}{a_{11}}$$

o

$$a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)}$$

dove il numero fra parentesi indica l'iterazione.

La procedura vista fin qui viene ripetuta per eliminare la prima incognita dalle equazioni rimanenti. Per esempio, la prima equazione normalizzata viene moltiplicata per a_{31} e il risultato viene sottratto dalla terza equazione per ottenere

$$a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + \cdots + a_{3n}^{(1)}x_n = b_3^{(1)}$$

Ripetendo queste operazioni su tutte le altre equazioni arriviamo al seguente sistema modificato:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 && (5.15)\\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} && (5.16)\\
a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + \cdots + a_{3n}^{(1)}x_n &= b_3^{(1)} && (5.17)\\
&\cdots\cdots\cdots\cdots\cdots\cdots\\
a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \cdots + a_{nn}^{(1)}x_n &= b_n^{(1)} && (5.18)
\end{aligned}$$

Nella procedura vista fin qui, la (5.11) viene detta *equazione pivot* e a_{11} viene detto *coefficiente pivot* o, semplicemente, *pivot*.

Ripetiamo quanto abbiamo appena fatto per eliminare, stavolta, la seconda incognita dalle equazioni (5.17)...(5.18). Per fare ciò, usiamo la (5.16) come equazione pivot e normalizziamola dividendola per il pivot $a_{22}^{(1)}$. Moltiplichiamo l'equazione normalizzata per $a_{32}^{(1)}$ e sottraiamo il risultato dalla (5.17) per eliminare la seconda incognita. Ripetiamo queste operazioni sulle rimanenti equazioni per ottenere

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + &\quad \ldots \quad + a_{1n}x_n &&= b_1\\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + &\quad \ldots \quad + a_{2n}^{(1)}x_n &&= b_2^{(1)}\\
a_{33}^{(2)}x_3 + &\quad \ldots \quad + a_{3n}^{(2)}x_n &&= b_3^{(2)}\\
&\quad \vdots &&\\
a_{n3}^{(2)}x_3 + &\quad \ldots \quad + a_{nn}^{(2)}x_n &&= b_n^{(2)}
\end{aligned}$$

dove il numero tra parentesi indica che il valore dei coefficienti, a questo punto, è stato modificato due volte.

Questa procedura viene poi ripetuta usando via via una delle equazioni rimaste come pivot. Il passo finale di tutta l'operazione consiste nell'usare la penultima equazione per eliminare la penultima incognita dall'ultima equazione. A questo punto, il sistema risulta trasformato in un sistema a matrice dei coefficienti triangolare superiore

$$\begin{array}{rcll} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + & \dots & + a_{1n}x_n & = b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + & \dots & + a_{2n}^{(1)}x_n & = b_2^{(1)} \\ a_{33}^{(2)}x_3 + & \dots & + a_{3n}^{(2)}x_n & = b_3^{(2)} \\ & & . & \\ & & . & \\ & & . & \\ & & a_{nn}^{(n-1)}x_n & = b_n^{(n-1)} \end{array} \tag{5.19}$$

Sostituzione all'indietro

Ora siamo in grado di risolvere l'ultima equazione del sistema (5.19) rispetto a x_n:

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \tag{5.20}$$

Questo risultato può essere sostituito nell'equazione precedente (ecco perché viene chiamata sostituzione all'indietro) per ricavare x_{n-1}. Questa procedura di sostituzione, che deve essere ripetuta per calcolare le rimanenti incognite, può essere rappresentata dalla seguente formula:

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \tag{5.21}$$

per $i = n-1, n-2, \dots, 1$.

Il metodo di eliminazione gaussiana necessita di $2(n-1)n(n+1)/3 + n(n-1)$ *flops*, a cui vanno aggiunti n^2 *flops* per risolvere il sistema triangolare con il metodo delle sostituzioni all'indietro. Servono dunque complessivamente circa $(2n^3/3 + 2n^2)$ *flops* per risolvere il sistema lineare. Più semplicemente, trascurando i termini di ordine inferiore, si può dire che il processo di eliminazione gaussiana costa $2n^3/3$ *flops*.

5.2.3 Programma per l'eliminazione gaussiana semplificata

La struttura generale di un programma per la risoluzione di un sistema di equazioni lineari algebriche con il metodo dell'eliminazione gaussiana, comprenderà quattro parti: lettura dei dati, eliminazione in avanti, sostituzione all'indietro e, infine, presentazione dei risultati. Qui di seguito proporremo una soluzione del corpo centrale di una procedura che realizza l'algoritmo di eliminazione gaussiana (in forma semplificata), lasciando al lettore il compito di gestire l'input dei dati e l'output dei risultati.

Ricordando che nell'algoritmo di eliminazione gaussiana occorre compiere le stesse operazioni su ogni riga di coefficienti e sul corrispondente termine noto, è conveniente aggiungere alla matrice dei coefficienti il vettore dei termini noti **b** e memorizzare l'insieme dei dati così composto nella matrice `A[n][n+1]`:

$$\mathtt{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n+1} \\ a_{21} & a_{22} & \dots & a_{2n+1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{n1} & a_{n2} & \dots & a_{nn+1} \end{bmatrix}$$

dove $[a_{1n+1} \dots a_{nn+1}]^T = \mathbf{b}$. In queste ipotesi e ricordando che gli indici dei vettori e delle matrici in "C" partono da zero, la k-esima iterazione della fase di eliminazione delle incognite può essere espressa dalla seguente relazione:

$$\mathtt{A[i][j]^{new}} = \mathtt{A[i][j]^{old}} - \mathtt{A[i][k]} * \mathtt{A[k][j]/A[k][k]}$$

con

$$\begin{aligned} \mathtt{k} &= 0, \dots, \mathtt{n}-2 \\ \mathtt{i} &= \mathtt{k}+1, \dots, \mathtt{n}-1 \\ \mathtt{j} &= \mathtt{k}, \dots, \mathtt{n} \end{aligned}$$

Per la fase di sostituzione in avanti è immediato utilizzare le relazioni (5.20) e (5.21).

Un esempio di frammento di codice per l'eliminazione gaussiana in forma semplificata può essere:

```
#define n 50

int i,j,k;                                        /* variabili di conteggio */
double A[n][n+1];              /* matrice dei coefficienti e termine noto */
double X[n];                       /* vettore delle incognite (risultato) */
double sum, tmp;                                    /* variabili temporanee */

                                                  /* eliminazione in avanti */
for(k=0;k<n-1;k++)
 {
  for(i=k+1;i<n;i++)
   {
    tmp=A[i][k];
    for(j=k;j<n+1;j++)
     A[i][j]=A[i][j]-(tmp/A[k][k])*A[k][j];
   }
```

```
 }

                                                /* sostituzione all'indietro */
X[n-1]=A[n-1][n]/A[n-1][n-1];
for(i=n-2;i>=0;i--)
 {
  sum=0.;
   for(j=i+1;j<n;j++)
    sum+=A[i][j]*X[j];
   X[i]=(A[i][n]-sum)/A[i][i];
 }
```

5.2.4 Punti deboli dei metodi di eliminazione

Nonostante il metodo di eliminazione gaussiana semplificata permetta di risolvere molti sistemi di equazioni diversi, esistono delle potenziali difficoltà che vanno prese in considerazione se si intende scrivere un programma di validità veramente generale. Quanto diremo nel seguito si riferisce in particolare all'eliminazione gaussiana semplificata, ma può essere ritenuto valido anche nel caso di altri metodi di eliminazione.

Divisione per zero

Il motivo principale per cui il metodo discusso finora viene definito semplificato è che esso non prende in considerazione la possibilità che possano essere tentate delle divisioni per zero. Infatti, se, per esempio, tentassimo di utilizzare l'eliminazione gaussiana semplificata per risolvere il sistema

$$\begin{array}{rcrcrcr} & & 2x_2 & + & 3x_3 & = & 8 \\ 4x_1 & + & 6x_2 & + & 7x_3 & = & -3 \\ 2x_1 & + & x_2 & + & 6x_3 & = & 5 \end{array}$$

la normalizzazione della prima equazione ci porterebbe ad eseguire una divisione per $a_{11} = 0$. Difficoltà di questo tipo sorgono anche quando un coefficiente è molto vicino a zero. Per evitare in parte questi problemi si può utilizzare la tecnica del *pivoting*, che verrà descritta nel Par. 5.2.5.

Errori di arrotondamento

Per evidenziare l'effetto degli errori di arrotondamento nell'eliminazione gaussiana vediamo un esempio. Sia dato il sistema

$$\begin{array}{rcrcrcr} 3x_1 & - & 0.1x_2 & - & 0.2x_3 & = & 7.85 \\ 0.1x_1 & + & 7x_2 & - & 0.3x_3 & = & -19.3 \\ 0.3x_1 & - & 0.2x_2 & + & 10x_3 & = & 71.4 \end{array}$$

la cui soluzione esatta è $x_1 = 3$, $x_2 = -2.5$, $x_3 = 7$. Risolvendolo col metodo dell'eliminazione gaussiana e utilizzando tre cifre significative durante lo svolgimento dei calcoli si ottiene, dopo l'eliminazione delle incognite, il seguente sistema triangolare:

$$\begin{array}{rcrcrcr} 3x_1 & - & 0.1x_2 & - & 0.2x_3 & = & 7.85 \\ & & 7.00x_2 & - & 0.293x_3 & = & -19.6 \\ & & & & 9.99x_3 & = & 70.1 \end{array}$$

La sostituzione all'indietro ci dà i valori delle incognite

$$\begin{aligned} x_1 &= 3.17 \\ x_2 &= -2.51 \\ x_3 &= 7.02 \end{aligned}$$

Sostituendo questi valori nelle equazioni originali si ottengono valori significativamente diversi da quelli reali

$$\begin{aligned} 8.37 &\neq 7.85 \\ -19.4 &\neq -19.3 \\ 71.7 &\neq 71.4 \end{aligned}$$

Anche se l'uso di sole tre cifre significative rende l'esempio poco realistico, bisogna tener presente che il problema dovuto all'arrotondamento esiste realmente e può diventare veramente importante quando le equazioni che compongono il sistema da risolvere sono molte: in tali casi, infatti, ogni risultato dipende da tutti i risultati precedenti; di conseguenza, un errore nelle prime operazioni tende a propagarsi, cioè a provocare errori nelle operazioni successive.

Stabilire per quali dimensioni del sistema gli errori di arrotondamento diventino importanti è difficile; il giudizio, poi, viene complicato dal fatto che il tipo di calcolatore e le caratteristiche delle equazioni sono fattori determinanti. Una regola approssimativa suggerisce che gli errori di arrotondamento diventano importanti quando si ha a che fare con sistemi di 25-50 equazioni. In ogni caso, è sempre bene sostituire i risultati trovati nelle equazioni originali per controllare che non si siano verificati sensibili errori. L'ordine di grandezza dei coefficienti può influenzare, però, l'affidabilità di un tale controllo.

Sistemi mal condizionati

In alcune situazioni la risoluzione di un sistema pone delle difficoltà. Per evidenziare queste circostanze ci avvaliamo di una interpretazione geometrica nel caso di due equazioni. Nella Fig. 5.2a le due equazioni rappresentano una coppia di rette parallele: in situazioni come questa non esiste soluzione in quanto le rette non si incrociano mai. In Fig. 5.2b le due rette sono coincidenti e, in questo caso, le soluzioni sono infinite. Entrambi questi tipi di sistemi vengono detti *singolari*. In generale, anche quando il sistema non è singolare, la bontà di una soluzione dipende da come è costituito il sistema in esame. In particolare, sistemi che hanno caratteristiche simili a quelle dei sistemi singolari possono dare luogo a delle difficoltà. Tali tipi di sistemi sono detti *mal condizionati*. Geometricamente, un sistema

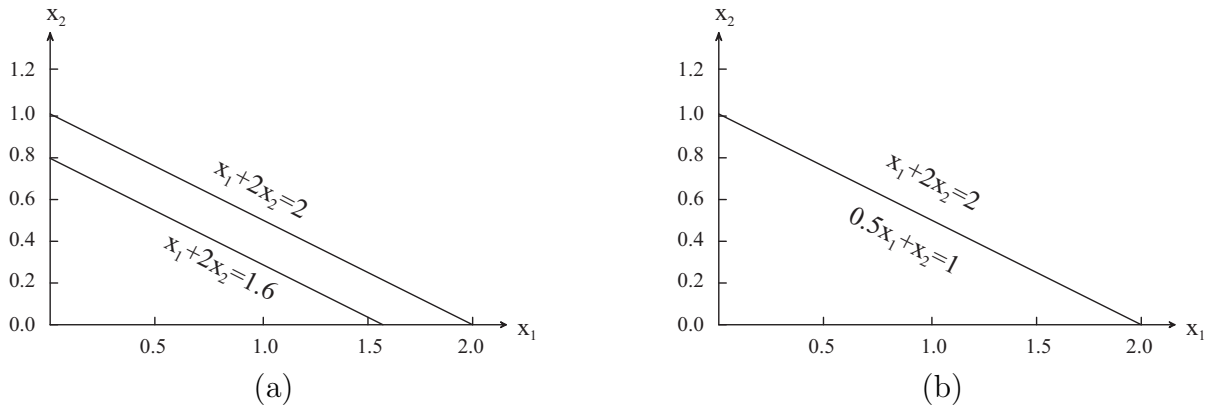


Figura 5.2: Rappresentazione grafica di sistemi singolari. (a) Nessuna soluzione. (b) Infinite soluzioni

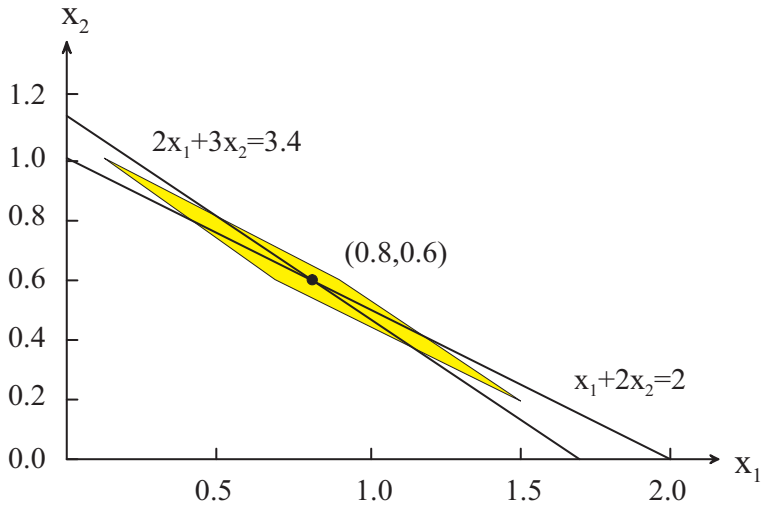


Figura 5.3: Rappresentazione grafica di un sistema mal condizionato. Le pendenze delle due rette sono così simili che il punto d'intersezione non può essere individuato visivamente con facilità. L'area ombreggiata rappresenta la regione in cui le due equazioni sono "quasi soddisfatte"

mal condizionato è rappresentato in Fig. 5.3 in cui non è facile visualizzare graficamente il punto esatto d'intersezione delle due rette.

In termini matematici, i *sistemi ben condizionati* sono quelli per i quali un piccolo cambiamento in uno o più coefficienti risulta in un cambiamento della stessa entità nella soluzione. Nei *sistemi mal condizionati*, invece, piccole variazioni nei coefficienti portano a grandi variazioni nella soluzione. Una interpretazione alternativa di mal condizionamento corrisponde al fatto che tutte le soluzioni contenute in un ampio campo di variabilità sono in grado di soddisfare approsimativamente il sistema di equazioni (v. regione ombreggiata in Fig. 5.3). Gli errori di arrotondamento portano proprio a piccole variazioni nei coefficienti che, a loro volta, possono portare a grandi differenze tra la soluzione trovata e quella esatta in sistemi mal condizionati. Una situazione di questo tipo viene illustrata nel seguente esempio.

Esempio Sia dato il sistema

$$\begin{array}{rcrcl} x_1 & + & 2x_2 & = & 10 \\ 1.1x_1 & + & 2x_2 & = & 10.4 \end{array}$$

Basandoci sulle equazioni (5.9) e (5.10) troviamo che la soluzione è:

$$x_1 = \frac{2 \cdot 10 - 2 \cdot 10.4}{1 \cdot 2 - 2 \cdot 1.1} = 4$$

$$x_2 = \frac{1 \cdot 10.4 - 1.1 \cdot 10}{1 \cdot 2 - 2 \cdot 1.1} = 3$$

Modificando leggermente il coefficiente di x_1 nella seconda equazione, da 1.1 a 1.05 e risolvendo il sistema, il risultato che troviamo è totalmente diverso:

$$x_1 = \frac{2 \cdot 10 - 2 \cdot 10.4}{1 \cdot 2 - 2 \cdot 1.05} = 8$$

$$x_2 = \frac{1 \cdot 10.4 - 1.05 \cdot 10}{1 \cdot 2 - 2 \cdot 1.05} = 1$$

Va notato che la differenza tra i due risultati è dovuta principalmente al fatto che il denominatore è costituito dalla differenza tra due numeri quasi uguali. Tali differenze risentono moltissimo di lievi variazioni negli operandi (vedi Cap. 1).

A questo punto si potrebbe pensare che la sostituzione dei risultati ottenuti nelle equazioni di partenza sia in grado di evidenziare il problema. Purtroppo, nel caso di sistemi mal condizionati non è così: sostituendo i valori errati $x_1 = 8$ e $x_2 = 1$ nelle equazioni del sistema otteniamo

$$\begin{array}{rcrcrcl} 8 & + & 2 \cdot 1 & = & 10 & = & 10 \\ 1.1 \cdot 8 & + & 2 \cdot 1 & = & 10.8 & \simeq & 10.4 \end{array}$$

Pertanto, anche se $x_1 = 8$ e $x_2 = 1$ non è la soluzione cercata per il sistema di partenza, il controllo dell'errore dà una risposta così vicina a quella esatta da riuscire a convincere l'utente che la soluzione ottenuta non è affetta da errore.

Il mal condizionamento del sistema risulta evidente quando si traccia il grafico delle rette. Poiché le pendenze delle due rette sono quasi identiche, è difficile individuare visivamente il punto di intersezione; questa difficoltà si riflette quantitativamente nei grandi errori di cui è affetta la soluzione del sistema. Questa situazione può essere caratterizzata matematicamente scrivendo le due equazioni nella forma generale:

$$a_{11}x_1 + a_{12}x_2 = b_1 \tag{5.22}$$
$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{5.23}$$

Dividendo la (5.22) per a_{12} e la (5.23) per a_{22} e riordinando le equazioni otteniamo una forma alternativa corrispondente alle equazioni di due rette $x_2 = (\text{pendenza})x_1 + \text{intercetta}$:

$$x_2 = -\frac{a_{11}}{a_{12}}x_1 + \frac{b_1}{a_{12}}$$

$$x_2 = -\frac{a_{21}}{a_{22}}x_1 + \frac{b_2}{a_{22}}$$

Quindi, se le due pendenze sono quasi uguali,

$$\frac{a_{11}}{a_{12}} \simeq \frac{a_{21}}{a_{22}}$$

ovvero,

$$a_{11}a_{22} - a_{21}a_{12} \simeq 0 \tag{5.24}$$

Ricordando che $a_{11}a_{22} - a_{21}a_{12}$ è il determinante della matrice dei coefficienti associata al sistema, giungiamo alla conclusione generale che tutti i sistemi mal condizionati hanno un determinante molto prossimo a zero. Nel caso limite, se il determinante fosse esattamente uguale a zero le due pendenze sarebbero identiche e il sistema non avrebbe alcuna soluzione oppure avrebbe infinite soluzioni, come nel caso dei sistemi singolari rappresentati graficamente in Fig. 5.2.

E' difficile specificare quanto vicino a zero debba essere il valore del determinante per indicare che il corrispondente sistema è mal condizionato; tale valutazione viene inoltre complicata dal fatto che il determinante cambia valore se una o più equazioni del sistema vengono moltiplicate per una costante (la soluzione, al contrario, in questo caso non cambia). Il valore del determinante, quindi, è relativo e può essere influenzato dall'ordine di grandezza dei coefficienti. L'ordine di grandezza dei coefficienti dà infatti luogo a un effetto di scala che complica la relazione tra il condizionamento di un sistema e il valore del determinante. Per superare in parte questa difficoltà è possibile normalizzare le equazioni in modo che l'elemento più grande di ogni riga della matrice dei coefficienti sia uguale a 1.

Come crediamo risulti chiaro da quanto detto sinora, non esiste un test semplice e univoco per determinare il grado di condizionamento del sistema. Fortunatamente, la maggior parte dei sistemi di equazioni lineari algebriche che derivano da problemi d'ingegneria sono già, per loro natura, ben condizionati. Inoltre, alcune delle tecniche descritte nel prossimo paragrafo aiutano a superare eventuali difficoltà dovute a sistemi mal condizionati.

Osservazione

La matrice inversa può essere utilizzata per riconoscere se un sistema è mal condizionato. I metodi adatti a questo scopo sono tre:

- Si normalizza la matrice dei coefficienti A in modo che l'elemento più grande di ogni riga sia uguale a 1. Se esistono elementi di A^{-1} di diversi ordini di grandezza superiori a quelli della matrice originale, è probabile che il sistema sia mal condizionato.

- Si moltiplica l'inversa per la matrice dei coefficienti originale e si verifica che la matrice risultante si discosti di poco dalla matrice unità. In caso contrario, il sistema analizzato è mal condizionato.

- Si inverte la matrice inversa e si controlla che il risultato si avvicini abbastanza alla matrice dei coefficienti originale. In caso contrario, il sistema è mal condizionato.

5.2.5 Miglioramento delle tecniche di risoluzione

Uso di un maggior numero di cifre significative

Il metodo più semplice per riuscire a trattare anche i sistemi mal condizionati consiste nell'impiegare un maggior numero di cifre significative. Se il calcolatore a vostra disposizione è in grado di manipolare numeri reali in doppia precisione, tale caratteristica aiuterà senz'altro a risolvere il problema.

Pivoting

Come accennato all'inizio del Par. 5.2.4, l'algoritmo di eliminazione gaussiana fallisce quando l'elemento pivot è uguale a zero: in questo caso, infatti, le operazioni di normalizzazione portano a tentare una divisione per zero. Anche se il pivot è prossimo a zero, ma non esattamente uguale a zero, possono sorgere delle difficoltà: infatti, se l'ordine di grandezza del pivot è molto minore di quello degli altri elementi, vengono introdotti errori di arrotondamento nei calcoli.

Prima di normalizzare ogni riga, quindi, vale la pena di determinare quale sia il coefficiente più grande (in valore assoluto) disponibile (nelle righe al di sotto di quella in esame e nella colonna a cui appartiene il pivot corrente); a questo punto, la riga corrente e quella che contiene nella colonna in esame il coefficiente più grande vengono scambiate di posto in modo che il pivot abbia il valore più grande possibile[3]. Questa tecnica viene detta *pivoting parziale*. Se la ricerca dell'elemento pivot e il successivo scambio avvengono sia secondo le righe che secondo le colonne della matrice dei coefficienti, si parla di *pivoting completo* (v. Fig. 5.4). Il pivoting completo viene usato raramente nei programmi più semplici in quanto la permutazione di colonne modifica l'ordine delle $\mathbf{x}$, aggiungendo così una sensibile e solitamente ingiustificata complessità al programma. Inoltre, va tenuto presente che,

[3]Se il pivot così ottenuto è nullo il sistema non è risolvibile

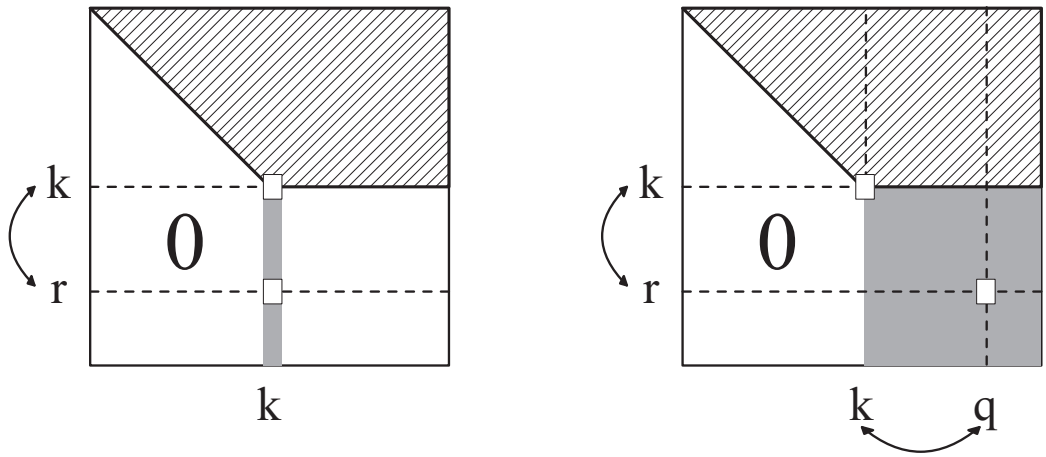


Figura 5.4: (a) Pivoting parziale (o per righe). (b) Pivoting totale. In grigio più scuro sono evidenziate le aree interessate alla ricerca dell'elemento pivotale

mentre la pivotazione parziale richiede un costo addizionale di circa n^2 confronti, quella completa ne richiede circa $2n^3/3$ con un considerevole aumento del costo computazionale dell'algoritmo di eleminazione gaussiana.

Il pivoting parziale, oltre ad evitare le divisioni per zero, permette di minimizzare gli errori di arrotondamento e, per questo motivo, contribuisce a superare le difficoltà dovute ad un mal condizionamento del sistema da risolvere[4]. I programmi di utilità generale, come MATLAB, spesso adottano una strategia di pivoting.

Tenendo conto della tecnica del pivoting parziale, il segmento di programma di eliminazione gaussiana presentato precedentemente può essere così modificato:

```
double temp;
#define Swap(x,y) {temp=(x); (x)=(y); (y)=temp;}

double pivot;
int ip;                                        /* indice di riga del pivot */

for(k=0;k<n-1;k++)                                  /* triangolarizzazione */
 {
  ricerco pivot in colonna k;
  if(pivot==0)
   {
    printf(Errore:  il sistema non ha soluzioni\n);
    exit(0);
```

[4]Non elimina il 'mal-condizionamento', ma ne contiente gli effetti negativi grazie ad errori di arrotondamento piú contenuti.

```
   }
  else if (ip!=k)
   {
    scambio riga k con riga ip;
    eseguo riduzione delle incognite riga k;
   }
 }

sostituzione all'indietro;
```

Sviluppiamo separatamente le operazioni indicate:

```
/* ricerco pivot in colonna k */
  ip=k;
  pivot=A[k][k];
  for(i=k+1;i<n;i++)
   {
    if(fabs(A[i][k])>fabs(pivot))
     {
      ip=i;
      pivot=A[i][k];
     }
   }


/* scambio riga k con riga ip */
   for(j=k;j<n+1;j++)
    Swap(A[k][j],A[ip][j]);


/* eseguo riduzione delle incognite riga k */
  for(i=k+1;i<n;i++)
   {
    tmp=A[i][k];
    for(j=k;j<n+1;j++)
     A[i][j]=A[i][j]-(tmp/A[k][k])*A[k][j];
   }
```

Il frammento di codice per l'eliminazione gaussiana con strategia di pivoting parziale risulta complessivamente:

```
#define n 50
double temp;
```

```
#define Swap(x,y) {temp=(x); (x)=(y); (y)=temp;}

int i,j,k,ip;
double A[n][n+1], X[n], sum, tmp, pivot;

for(k=0;k<n-1;k++)                                  /* triangolarizzazione */
 {
  ip=k;                                               /* fase di pivoting */
  pivot=A[k][k];
  for(i=k+1;i<n;i++)
   {
    if(fabs(A[i][k])>fabs(pivot))
     {
      ip=i;
      pivot=A[i][k];
     }
   }                                               /* fine fase di pivoting */
  if(pivot==0)
   {
    printf(Errore:  il sistema non ha soluzioni\n);
    exit(0);
   }
  else if (ip!=k)
   for(j=k;j<n+1;j++)
    Swap(A[k][j],A[ip][j]);                                 /* scambio riga */
  for(i=k+1;i<n;i++)                               /* eliminazione in avanti */
   {
    tmp=A[i][k];
    for(j=k;j<n+1;j++)
     A[i][j]=A[i][j]-(tmp/A[k][k])*A[k][j];
   }                                           /* fine eliminazione in avanti */
 }                                                /* fine triangolarizzazione */

 X[n-1]=A[n-1][n]/A[n-1][n-1];                 /* sostituzione all'indietro */
 for(i=n-2;i>=0;i--)
  {
   sum=0.;
   for(j=i+1;j<n;j++)
    sum+=A[i][j]*X[j];
   X[i]=(A[i][n]-sum)/A[i][i];
  }                                       /* fine sostituzione all'indietro */
```

Normalizzazione

La normalizzazione consente di minimizzare gli errori di arrotondamento nei casi in cui alcune delle equazioni hanno coefficienti molto maggiori delle altre. Situazioni di questo tipo si incontrano frequentemente in ingegneria, quando nello sviluppo del sistema di equazioni vengono utilizzate unità di misura molto diverse; per esempio, nell'analisi dei circuiti elettrici le tensioni incognite possono essere espresse in unità che vanno dai microvolt (μV) ai chilovolt (kV). Esempi simili si possono trovare in tutti i campi dell'ingegneria. Se le unità utilizzate in ogni equazione sono tra loro coerenti e dello stesso ordine di grandezza, il sistema sarà fisicamente corretto e risolvibile. L'uso di unità con ordini di grandezza diversi, invece, porta a coefficienti molto diversi tra loro; ciò, a sua volta, può aumentare gli errori di arrotondamento in quanto tende a influenzare il pivoting.

Sebbene la normalizzazione permetta di ridurre gli errori di arrotondamento, va notato che comporta essa stessa un arrotondamento. Per esempio, data l'equazione

$$2x_1 \quad + \quad 300000x_2 \quad = \quad 1$$

e mantenendo tre cifre significative, la normalizzazione dà

$$0.00000667x_1 \quad + \quad x_2 \quad = \quad 0.00000333$$

La normalizzazione, quindi, introduce un errore di arrotondamento nel primo coefficiente e nel termine noto; per questo motivo, vale la pena di *eseguire la normalizzazione solo quando ce n'è effettivamente bisogno*, cioè quando nel sistema da risolvere sono presenti coefficienti di ordini di grandezza molto diversi.

5.2.6 Calcolo del determinante per mezzo dell'eliminazione gaussiana

Il metodo si basa sul fatto che il determinante di una matrice triangolare può essere calcolato come prodotto degli elementi della diagonale

$$\det(A) = a_{11}a_{22}a_{33}\cdots a_{nn}$$

Ricordando che la fase di eliminazione in avanti del metodo di eliminazione gaussiana dà come risultato un sistema con matrice dei coefficienti triangolare superiore, e poiché il valore del determinante non viene modificato dal processo di eliminazione in avanti, il valore cercato può essere calcolato alla fine di questa fase per mezzo della formula

$$\det(A) = a_{11}a_{22}^{(1)}a_{33}^{(2)}\cdots a_{nn}^{(n-1)} \tag{5.25}$$

Quindi, con una sola operazione, la riduzione della matrice dei coefficienti a forma triangolare superiore, otteniamo due risultati: un sistema di risoluzione immediata e un'altrettanto immediata stima del valore del determinante.

Se il programma di risoluzione dei sistemi lineari implementa il pivoting parziale, il metodo appena visto richiede una piccola modifica in quanto il determinante cambia segno ogni volta che avviene una permutazione di righe. Per tener conto di questo fatto, basta modificare come segue la (5.25):

$$\det(A) = a_{11}a_{22}^{(1)}a_{33}^{(2)} \cdots a_{nn}^{(n-1)}(-1)^p \tag{5.26}$$

dove p rappresenta il numero di permutazioni effettuate. L'introduzione nel programma di questa modifica non presenta difficoltà: basta contare le permutazioni durante l'esecuzione del programma e utilizzare poi la (5.26) per calcolare il determinante.

5.3 Metodo di Gauss-Jordan

Il metodo di Gauss-Jordan è una variante dell'eliminazione gaussiana; la differenza principale è che, nel metodo di Gauss-Jordan, quando si elimina una variabile la si elimina da tutte le equazioni del sistema e non solo da quelle che stanno sotto alla riga corrente: pertanto, la fase di eliminazione dà come risultato una matrice unità anziché una matrice triangolare superiore. Non è, quindi, necessario eseguire la sostituzione all'indietro per arrivare alla soluzione. Vediamo un esempio.

Supponiamo di voler risolvere il seguente sistema con il metodo di Gauss-Jordan

$$\begin{array}{rcrcrcr} 3x_1 & - & 0.1x_2 & - & 0.2x_3 & = & 7.85 \\ 0.1x_1 & + & 7x_2 & - & 0.3x_3 & = & -19.3 \\ 0.3x_1 & - & 0.2x_2 & + & 10x_3 & = & 71.4 \end{array}$$

Per prima cosa, rappresentiamo la matrice dei coefficienti e il vettore dei termini noti come un'unica matrice rettangolare:

$$\left[\begin{array}{cccc} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$$

Normalizziamo poi la prima riga dividendola per il pivot (3):

$$\left[\begin{array}{cccc} 1 & -0.0333333 & -0.0666667 & 2.61667 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$$

L'incognita x_1 viene eliminata dalla seconda riga sottraendo la prima riga moltiplicata per 0.1 dalla seconda. Allo stesso modo, sottraendo la prima riga moltiplicata per 0.3 dalla terza si elimina x_1 dalla terza riga:

$$\left[\begin{array}{cccc} 1 & -0.0333333 & -0.0666667 & 2.61667 \\ 0 & 7.00333 & -0.29333 & -19.5617 \\ 0 & -0.190000 & 10.0200 & 70.6150 \end{array}\right]$$

Normalizziamo ora la seconda riga dividendola per 7.00333:

$$\left[\begin{array}{cccc} 1 & -0.0333333 & -0.0666667 & 2.61667 \\ 0 & 1 & -0.0418848 & -2.79320 \\ 0 & -0.190000 & 10.0200 & 70.6150 \end{array}\right]$$

Eliminando x_2 dalla prima e dalla terza riga, otteniamo:

$$\left[\begin{array}{cccc} 1 & 0 & -0.0680629 & 2.52356 \\ 0 & 1 & -0.0418848 & -2.79320 \\ 0 & 0 & 10.0120 & 70.0843 \end{array}\right]$$

Normalizziamo ora la terza riga dividendola per 10.0120:

$$\left[\begin{array}{cccc} 1 & 0 & -0.0680629 & 2.52356 \\ 0 & 1 & -0.0418848 & -2.79320 \\ 0 & 0 & 1 & 7.00003 \end{array}\right]$$

Infine, eliminando x_3 dalla prima e dalla seconda riga; otteniamo:

$$\left[\begin{array}{cccc} 1 & 0 & 0 & 3.00000 \\ 0 & 1 & 0 & -2.50001 \\ 0 & 0 & 1 & 7.00003 \end{array}\right]$$

La matrice dei coefficienti è stata trasformata nella matrice unità e la soluzione si trova nel vettore dei termini noti. Pertanto, non è necessario eseguire la sostituzione all'indietro per ottenere la soluzione.

Tutto quanto detto riguardo ai punti deboli e a i possibili miglioramenti dell'eliminazione gaussiana vale anche per il metodo di Gauss-Jordan; per esempio, per evitare le divisioni per zero e per limitare gli errori di arrotondamento è possibile applicare una strategia di pivoting simile a quella già vista.

Dal punto di vista del costo computazionale, il metodo di Gauss-Jordan e l'eliminazione gaussiana possono sembrare molto simili; in realtà il metodo di Gauss-Jordan richiede circa il 50% di operazioni in più. Per questo motivo, l'eliminazione gaussiana è il metodo diretto preferito per la soluzione di sistemi di equazioni lineari. Il nostro interesse per il metodo di Gauss-Jordan è tuttavia dovuto al fatto che questa tecnica permette di calcolare numericamente l'inversa di una matrice in modo semplice ed efficiente, come verrà descritto nel Par.5.3.2.

5.3.1 Programma per il metodo di Gauss-Jordan

Presentiamo un segmento di codice "C" che implementa l'algoritmo di Gauss-Jordan senza pivoting parziale, osservando che è possibile incorporare nel codice uno schema di pivoting simile a quello presentato per l'eliminazione gaussiana.

Utilizzando la matrice `A[n][n+1]` per memorizzare la matrice ottenuta aggiungendo la colonna del termine noto alla matrice dei coefficienti, l'algoritmo di Gauss-Jordan può essere così riassunto:

$$\mathtt{A[k][j]^{new}} = \mathtt{A[k][j]^{old}} / \mathtt{A[k][k]}$$

con

$$\begin{aligned} \mathtt{k} &= 0, \ldots, \mathtt{n}-1 \\ \mathtt{j} &= \mathtt{k}, \ldots, \mathtt{n} \end{aligned}$$

e

$$\mathtt{A[i][j]^{new}} = \mathtt{A[i][j]^{old}} - \mathtt{A[k][j]} * \mathtt{A[i][k]}$$

con

$$\begin{aligned} \mathtt{k} &= 0, \ldots, \mathtt{n}-1 \\ \mathtt{i} &= 0, \ldots, \mathtt{n}-1 \qquad \mathtt{i} \neq \mathtt{k} \\ \mathtt{j} &= \mathtt{k}, \ldots, \mathtt{n} \end{aligned}$$

Un esempio di frammento di codice che implementa l'algoritmo può essere

```
#define n 50

int i,j,k;
double A[n][n+1];
double sum,tmp1,tmp2;

for(k=0;k<n;k++)
 {
  for(i=0;i<n;i++)
   {
    if(i!=k)
     {
      tmp1=A[k][k];
      tmp2=A[i][k];
      for(j=k;j<n+1;j++)
       {
        A[k][j]=A[k][j]/tmp1;
        A[i][j]=A[i][j]-tmp2*A[k][j];
       }
     }
   }
 }
```

Al termine delle (n) iterazioni, la soluzione sarà contenuta nelle variabili

$$\mathtt{A[0][n]}, \mathtt{A[1][n]}, \ldots, \mathtt{A[n-1][n]} \ .$$

5.3.2 Calcolo dell'inversa

Un caso in cui l'inversa trova applicazione riguarda la risoluzione di diversi sistemi di equazioni nella forma

$$A\mathbf{x} = \mathbf{b}$$

quando questi sistemi differiscano solo per il vettore dei termini noti $\mathbf{b}$. In questo caso, invece di risolvere i singoli sistemi separatamente, si adotta un approccio diverso: si calcola una sola volta l'inversa della matrice dei coefficienti; poi, per mezzo della relazione

$$\mathbf{x} = A^{-1}\mathbf{b}$$

si ottiene la soluzione di ogni sistema moltiplicando A^{-1} per il particolare vettore $\mathbf{b}$ d'interesse[5]. Dato che la moltiplicazione tra matrici è molto più veloce dell'inversione, è preferibile eseguire la parte laboriosa dei calcoli una sola volta per poi ottenere velocemente le varie soluzioni.

Per il calcolo dell'inversa ci si può avvalere del metodo di Gauss-Jordan. A questo scopo, alla matrice dei coefficienti viene aggiunta una matrice unità (v. Fig. 5.5); in seguito, il metodo di Gauss-Jordan riduce la matrice dei coefficienti a una matrice unità. A questo punto, la metà di destra della matrice conterrà l'inversa della matrice originale.

E' immediato modificare il programma presentato precedentemente, che implementa il metodo di Gauss-Jordan, in modo da consentire il calcolo della matrice inversa. A questo scopo, alla matrice dei coefficienti va aggiunta una matrice unità all'inizio del programma; inoltre alcuni dei contatori dei cicli devono essere aumentati in modo che i calcoli vengano eseguiti su tutte le colonne della matrice aggiunta.

Se il programma per il metodo di Gauss-Jordan prevedeva il pivoting parziale, si rendono necessarie modifiche più profonde: infatti, se due righe della matrice dei coefficienti vengono permutate, anche le corrispondenti colonne della matrice inversa devono subire lo stesso spostamento. La Fig. 5.6 illustra questa operazione. Se, per esempio, le posizioni delle righe 2 e 3 vengono scambiate, il "significato" o "interpretazione" delle colonne 2 e 3 cambia: ciò che prima rappresentava l'effetto sulle $\mathbf{x}$ di una variazione unitaria di b_2 ora indica l'effetto di una variazione unitaria di b_3, e viceversa.

Oltre a quanto appena stabilito, il programma deve essere in grado di calcolare le soluzioni del sistema per un numero arbitrario di vettori dei termini noti. Ciò si può ottenere semplicemente aggiungendo un ciclo al termine del calcolo dell'inversa; all'interno di questo ciclo il programma richiede all'utente un nuovo vettore dei termini noti che andrà moltiplicato per A^{-1} per ottenere una nuova soluzione. La procedura viene così ripetuta fino a quando l'utente non comunica al programma che non sono richieste altre soluzioni.

[5] Anche con il metodo di eliminazione gaussiana è possibile risolvere diversi sistemi che differiscano solo per il vettore dei termini noti applicando il metodo ad una matrice A di n righe e $m = n + p$ colonne, contenente in ciascuna colonna dalla $(n+1)$-esima alla $(n+p)$-esima un vettore di termini noti. Le operazioni per l'eliminazione delle incognite verranno estese a valori di $j = 1, \ldots, m$. In questo caso occorre, tuttavia, conoscere a priori (cioè prima di poter applicare il metodo di eliminazione gaussiana) tutti i p vettori dei termini noti dei sistemi da risolvere.

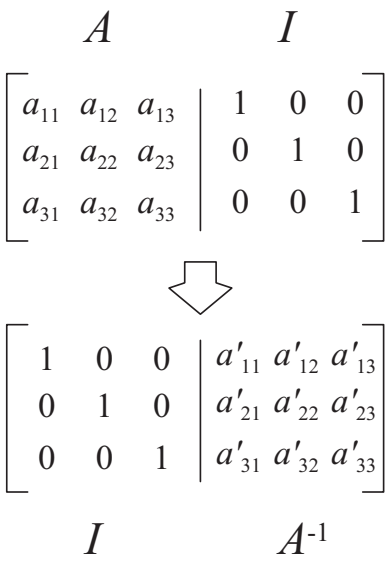


Figura 5.5: Metodo di Gauss-Jordan con inversione della matrice. I coefficienti a'_{ij} identificano gli elementi della matrice inversa A^{-1}

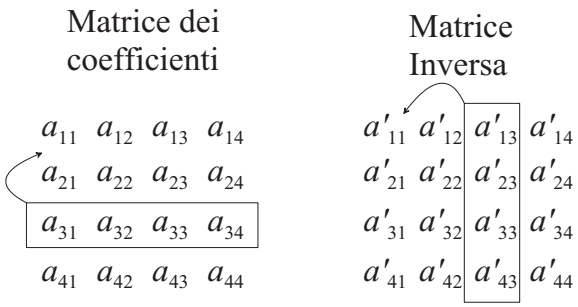


Figura 5.6: Lo spostamento di una riga della matrice dei coefficienti (nel pivoting parziale) influisce sul posizionamento delle colonne della matrice inversa risultante

5.4 Fattorizzazione LU

Supponiamo di poter scrivere la matrice A come il prodotto di due matrici

$$A = LU \tag{5.27}$$

con L matrice triangolare inferiore e U matrice triangolare superiore. Nel caso di una matrice A 4×4, ad esempio, l'equazione (5.27) apparirebbe come:

$$\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

La *fattorizzazione* o *decomposizione* LU può essere utilizzata per riformulare la soluzione di un sistema di equazioni lineari

$$A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b} \tag{5.28}$$

risolvendo dapprima, rispetto a $\mathbf{y}$ il sistema

$$L\mathbf{y} = \mathbf{b} \tag{5.29}$$

e poi, rispetto a $\mathbf{x}$ il sistema

$$U\mathbf{x} = \mathbf{y} \tag{5.30}$$

Pertanto, una volta note le matrici L e U, la risoluzione del sistema lineare di partenza comporta la risoluzione (successiva) di due sistemi triangolari, col vantaggio che la soluzione di un sistema triangolare è abbastanza banale in quanto è sufficiente effettuare una sostituzione delle incognite. In particolare, la (5.29) può essere risolta mediante una sostituzione in avanti:

$$\begin{aligned} y_1 &= \frac{b_1}{l_{11}} \\ y_i &= \frac{1}{l_{ii}} \left[b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right] \qquad i = 2, 3, \ldots, n \end{aligned}$$

mentre la (5.30) può essere risolta mediante una sostituzione all'indietro:

$$\begin{aligned} x_n &= \frac{y_n}{u_{nn}} \\ x_i &= \frac{1}{u_{ii}} \left[y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right] \qquad i = n-1, n-2, \ldots, 1 \end{aligned}$$

Il costo computationale di queste operazioni è pari a n^2 *flops* a cui si deve però aggiungere il costo della fattorizzazione LU. Complessivamente il costo della tecnica di fattorizzazione LU è uguale a quello dell'eliminazione gaussiana.

Si preferisce ricorrere alla fattorizzazione LU quando si deve risolvere il sistema $A\mathbf{x} = \mathbf{b}$ per diversi valori di $\mathbf{b}$, in questi casi il metodo risulta conveniente perché la fattorizzazione viene effettuata una volta sola.

5.4.1 Forme compatte di fattorizzazione

Il calcolo di una fattorizzazione LU di A equivale formalmente alla risoluzione del sistema non lineare di n^2 equazioni

$$a_{ij} = \sum_{r=1}^{\min(i,j)} l_{ir} u_{rj} \tag{5.31}$$

in cui le incognite sono gli $n^2 + n$ coefficienti (non nulli) delle matrici triangolari L e U. Poiché abbiamo più incognite che equazioni possiamo fissare arbitrariamente n coefficienti uguali a 1, ad esempio gli elementi diagonali di L o quelli di U. In tal modo si ottengono, rispettivamante, i metodi di Doolittle e di Crout, che risolvono in maniera efficiente il sistema (5.31). Supponendo di conoscere infatti le prime $k-1$ colonne di L e di U e ponendo $l_{kk} = 1$ (metodo Doolittle), abbiamo dalla (5.31) le equazioni

$$\begin{aligned} a_{kj} &= \sum_{r=1}^{k-1} l_{kr} u_{rj} + u_{kj} && j = k, \ldots, n \\ a_{ik} &= \sum_{r=1}^{k-1} l_{ir} u_{rk} + l_{ik} u_{kk} && i = k+1, \ldots, n \end{aligned}$$

che possono essere quindi risolte in maniera *sequenziale* nelle incognite u_{rj} e l_{ir}. Di conseguenza, con le formule compatte del metodo di Doolittle, si ottiene sequenzialmente prima la riga k-esima di U e poi la colonna k-esima di L, come segue, per $k = 1, \ldots, n$

$$\begin{aligned} u_{kj} &= a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj} && j = k, \ldots, n \\ l_{ik} &= \frac{1}{u_{kk}} \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk} \right) && i = k+1, \ldots, n \end{aligned}$$

In maniera analoga si ottiene la fattorizzazione di Crout calcolando in sequenza prima la colonna k-esima di L, poi la riga k-esima di U, per $k = 1, \ldots, n$:

$$\begin{aligned} l_{ik} &= a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk} && i = k, \ldots, n \\ u_{kj} &= \frac{1}{l_{kk}} \left(a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj} \right) && j = k+1, \ldots, n \, . \end{aligned}$$

Osservazione Dal punto di vista implementativo, essendo L triangolare inferiore con elementi noti a priori sulla diagonale principale (sono pari ad 1) ed U triangolare superiore, è possibile memorizzare la fattorizzazione LU direttamente nella stessa area di memoria occupata dalla matrice A. Precisamente U occupa la parte triangolare superiore di A (diagonale principale inclusa), mentre L la parte triangolare inferiore (gli elementi diagonali di L non sono memorizzati essendo esplicitamente assunti pari a 1).

5.5 Raffinamento iterativo

In alcuni casi, i metodi diretti (nonostante l'adozione di speciali accorgimenti, quali ad esempio il pivoting parziale e la normalizzazione) non sono sufficienti ad assicurare risultati precisi. Questa imprecisione, dovuta alla propagazione degli errori di arrotondamento, può essere ridotta attraverso una compensazione iterativa degli errori che porta ad un progressivo raffinamento della soluzione. Consideriamo un sistema di equazioni lineari

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & = & b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & = & b_n \end{array} \tag{5.32}$$

Supponiamo di aver ottenuto un vettore soluzione approssimato, dato da $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$. Questi risultati, sostituiti nel sistema (5.32), danno

$$\begin{array}{rcl} a_{11}\tilde{x}_1 + a_{12}\tilde{x}_2 + \cdots + a_{1n}\tilde{x}_n & = & \tilde{b}_1 \\ a_{21}\tilde{x}_1 + a_{22}\tilde{x}_2 + \cdots + a_{2n}\tilde{x}_n & = & \tilde{b}_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & \\ a_{n1}\tilde{x}_1 + a_{n2}\tilde{x}_2 + \cdots + a_{nn}\tilde{x}_n & = & \tilde{b}_n \end{array} \tag{5.33}$$

Supponiamo che la soluzione esatta $x_1, x_2, \ldots, x_n$ sia espressa in funzione di $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ e dei fattori di correzione $\delta_1, \delta_2, \ldots, \delta_n$:

$$\begin{array}{rcl} x_1 = \tilde{x}_1 + \delta_1 \\ x_2 = \tilde{x}_2 + \delta_2 \\ \quad\cdot\qquad\cdot \\ \quad\cdot\qquad\cdot \\ \quad\cdot\qquad\cdot \\ x_n = \tilde{x}_n + \delta_n \end{array} \tag{5.34}$$

Se queste uguaglianze vengono sostituite nella (5.32) si ottiene il seguente sistema:

$$\begin{array}{rcl} a_{11}(\tilde{x}_1 + \delta_1) + a_{12}(\tilde{x}_2 + \delta_2) + \cdots + a_{1n}(\tilde{x}_n + \delta_n) & = & b_1 \\ a_{21}(\tilde{x}_1 + \delta_1) + a_{22}(\tilde{x}_2 + \delta_2) + \cdots + a_{2n}(\tilde{x}_n + \delta_n) & = & b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & \\ a_{n1}(\tilde{x}_1 + \delta_1) + a_{n2}(\tilde{x}_2 + \delta_2) + \cdots + a_{nn}(\tilde{x}_n + \delta_n) & = & b_n \end{array} \tag{5.35}$$

Ora, sottraendo la (5.33) dalla (5.35), otteniamo

$$\begin{array}{rcccl} a_{11}\delta_1 + a_{12}\delta_2 + \cdots + a_{1n}\delta_n & = & b_1 - \tilde{b}_1 & = & r_1 \\ a_{21}\delta_1 + a_{22}\delta_2 + \cdots + a_{2n}\delta_n & = & b_2 - \tilde{b}_2 & = & r_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & & \\ a_{n1}\delta_1 + a_{n2}\delta_2 + \cdots + a_{nn}\delta_n & = & b_n - \tilde{b}_n & = & r_n \end{array} \tag{5.36}$$

Abbiamo così un nuovo sistema di equazioni lineari la cui soluzione è data dai fattori di correzione. Una volta calcolati, i fattori di correzione possono essere sostituiti nella (5.34) per ottenere una migliore soluzione del sistema originale.

In generale se $\mathbf{x}^{(0)}$ è la soluzione calcolata con un metodo diretto, fissata una certa tolleranza sull'errore, ε, il raffinamento iterativo consiste nei seguenti passi (per $i = 0, 1, \ldots$ fino a convergenza):

1. calcolare il residuo $\mathbf{r}^{(i)} = \mathbf{b} - A\mathbf{x}^{(i)}$;
2. risolvere il sistema lineare $A\boldsymbol{\delta} = \mathbf{r}^{(i)}$ ([6]);
3. aggiornare la soluzione ponendo $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \boldsymbol{\delta}$;
4. se $\|\boldsymbol{\delta}\|/\|\mathbf{x}^{(i+1)}\| < \varepsilon$, terminare il processo prendendo come soluzione $\mathbf{x}^{(i+1)}$. In caso contrario, l'algoritmo riparte dal punto 1.

In assenza di errori di arrotondamento, il processo fornirebbe la soluzione esatta dopo un solo passo. Le proprietà di convergenza del metodo possono essere migliorate calcolando il solo residuo $\mathbf{r}^{(i)}$ con una precisione doppia rispetto a tutte le altre quantità. C'è comunque convenienza nell'uso del raffinamento iterativo, anche a precisione fissata. In generale, infatti, un qualunque metodo diretto accoppiato con il raffinamento iterativo guadagna in stabilità.

In conclusione, il raffinamento iterativo è una tecnica che consente di migliorare l'accurattezza della soluzione ottenuta con un metodo diretto. Più precisamente quando il sistema non è troppo mal condizionato il processo di raffinamento elimina l'eventuale propagazione di errori dovuta ad una possibile non perfetta stabilità dell'algoritmo di Gauss (ma non elimina la propagazione dovuta al condizionamento del problema).

5.6 Metodi di Jacobi e di Gauss-Seidel

5.6.1 Generalità sui metodi iterativi

I metodi iterativi si contrappongono ai metodi diretti perché la soluzione $\mathbf{x}$ del sistema lineare $A\mathbf{x} = \mathbf{b}$ viene ottenuta formalmente dopo un numero infinito di passi. Nel caso di matrici piene (con la maggior parte dei coefficienti diversi da zero), il costo computazionale da essi richiesto è dell'ordine di n^2 operazioni per ogni iterazione, rispetto al costo dell'ordine di n^3 operazioni tipicamente richiesto dai metodi diretti per fornire l'intera soluzione. Sarà dunque necessario disporre di metodi iterativi rapidamente convergenti, in modo da mantenere al di sotto di n^3 il costo computazionale globale. Se si deve risolvere un solo sistema con n grande o con matrici sparse, allora i metodi iterativi risulteranno in generale preferibili ai metodi diretti. Inoltre essi comportano una ridotta occupazione di memoria.

[6] Dal momento che il raffinamento iterativo necessita la risoluzione di diversi sistemi con la stessa matrice dei coefficienti A, è conveniente utilizzare metodi quali la fattorizzazione LU o il metodo di Gauss-Jordan, in modo da poter riutilizzare la fattorizzazione o l'inversa precedentemente calcolata.

I metodi iterativi sono adatti a risolvere sistemi in cui gli errori di arrotondamento diventano importanti; infatti, con tali metodi è possibile iterare la procedura di calcolo finché la stima della soluzione non converge sul valore esatto entro un predeterminato margine di errore. L'errore di arrotondamento, così, non ha più rilevanza, dato che il livello ammissibile dell'errore è sotto il controllo dell'utente. Formalmente, si basano sull'idea di calcolare una successione di vettori $\mathbf{x}^{(k)}$ che godano della proprietà di convergenza

$$\mathbf{x} = \lim_{k \to \infty} \mathbf{x}^{(k)} ,$$

essendo $\mathbf{x}$ la soluzione di $A\mathbf{x} = \mathbf{b}$. Naturalmente, ci si vorrebbe fermare al minimo k per cui di abbia $\|\mathbf{x}^{(k)} - \mathbf{x}\| < \varepsilon$, dove ε è una tolleranza fissata a priori e $\| \cdot \|$ è un'opportuna norma vettoriale, ad esempio euclidea (v. App. A). Tuttavia, non essendo disponibile la soluzione esatta, sarà necessario introdurre degli opportuni criteri d'arresto basati su altre quantità. Rimandiamo al Par. 5.6.5 per una discussione sull'argomento.

I metodi che tratteremo nel seguito appartengono ad un'ampia classe di metodi iterativi esprimibili nella forma:

$$\mathbf{x}^{(0)} \quad \text{dato}, \qquad \mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{f} , \qquad k \geq 0 .$$

dove M è una matrice quadrata $n \times n$ detta *matrice di iterazione* e $\mathbf{f}$ è un vettore che si ottiene a partire dal termine noto $\mathbf{b}$. In particolare, presenteremo due "classici" metodi iterativi lineari, basati su una decomposizione additiva della matrice dei coefficienti A.

5.6.2 Tecniche di disaccoppiamento

Supponiamo di scomporre la matrice A in

$$A = L + D + U$$

come indicato in Fig. 5.7.

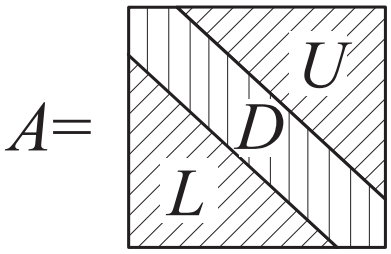


Figura 5.7: Decomposizione della matrice.

In questo modo, dal sistema originario $A\mathbf{x} = \mathbf{b}$ si può scrivere il sistema equivalente

$$D\mathbf{x} = -(L + U)\mathbf{x} + \mathbf{b}$$

Eliminando i vincoli di interazione esistenti tra le equazioni del sistema, queste possono essere disaccoppiate e la soluzione può essere determinata per via iterativa, utilizzando la soluzione approssimata al passo k-esimo, ottenuta dal sistema disaccoppiato (cioè risolvendo n equazioni distinte), per determinare la soluzione approssimata al passo $(k+1)$-esimo:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= -D^{-1}[(L+U)\mathbf{x}^{(k)} - \mathbf{b}] \\ \mathbf{x}^{(k+1)} &= M_J \mathbf{x}^{(k)} + \mathbf{f}_J \end{aligned} \tag{5.37}$$

dove $M_J = -D^{-1}(L+U)$ è la matrice di iterazione di Jacobi e $\mathbf{f}_J = D^{-1}\mathbf{b}$.

Partendo dalla stessa decomposizione additiva della matrice dei coefficienti A, e mettendo in evidenza i termini in modo leggermente diverso da prima si ottiene un altro sistema equivalente a quello originario

$$(L+D)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$$

Anche in questo caso si può introdurre un procedimento iterativo che disaccoppia le equazioni del sistema e permette di ottenere una succesione di soluzioni approssimate:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= -(L+D)^{-1}[U\mathbf{x}^{(k)} - \mathbf{b}] \\ \mathbf{x}^{(k+1)} &= M_{GS} \mathbf{x}^{(k)} + \mathbf{f}_{GS} \end{aligned} \tag{5.38}$$

dove $M_{GS} = -(L+D)^{-1}U$ è la matrice di iterazione di Gauss-Seidel e $\mathbf{f}_{GS} = (L+D)^{-1}\mathbf{b}$.

5.6.3 Metodo di Jacobi

Il metodo di Jacobi si basa sul diretto utilizzo della relazione (5.37) per cui, scelto una soluzione iniziale arbitraria $\mathbf{x}^{(0)}$, si calcola $\mathbf{x}^{(k+1)}$ attraverso la formula

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right] , \qquad i = 1, \ldots, n \tag{5.39}$$

Supponiamo di avere un sistema di n equazioni $A\mathbf{x} = \mathbf{b}$, se gli elementi della diagonale principale sono tutti diversi da zero, è possibile risolvere la prima equazione rispetto a x_1, la seconda rispetto a x_2 e così via fino ad ottenere

$$\begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)/a_{11} \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2 - \cdots - a_{3n}x_n)/a_{33} \\ \cdot \quad &\cdot \qquad\qquad\qquad \cdot \quad \cdot \quad \cdot \\ x_n &= (b_n - a_{n2}x_2 - a_{n3}x_3 - \cdots - a_{nn-1}x_{n-1})/a_{nn} \ . \end{aligned} \tag{5.40}$$

Il procedimento di risoluzione comincia con la scelta dei valori iniziali per le $\mathbf{x}$; nell'ipotesi più semplice, si assume che tutte le $\mathbf{x}$ siano uguali a zero. I valori iniziali vengono sostituiti nelle Eqq. (5.40) per calcolare un intero vettore di nuove $\mathbf{x}$ in base ai vecchi valori delle $\mathbf{x}$ stesse. A questo punto, si ritorna alla prima equazione per ripetere da capo tutto il procedimento fino a quando la soluzione converge ai valori esatti entro i limiti prefissati.

5.6.4 Metodo di Gauss-Seidel

Il metodo di Gauss-Seidel è il metodo iterativo più comunemente usato e si basa su una generalizzazione della (5.39). Esso si differenzia dal metodo di Jacobi per il fatto che al passo $(k+1)$-esimo si utilizzano i valori di $x_i^{(k+1)}$ qualora siano disponibili. Pertanto, invece della (5.39), si usa la formula seguente

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right] , \qquad i = 1, \ldots, n \tag{5.41}$$

Nel metodo di Gauss-Seidel, quindi, il nuovo valore di ogni incognita viene immediatamente utilizzato nell'equazione successiva per calcolare il nuovo valore di un'altra incognita; in tal modo, se i risultati convergono alla soluzione, ogni operazione fa uso della migliore stima disponibile.

Esempio Sia dato il sistema

$$\begin{array}{rcrcrcl} 3x_1 & - & 0.1x_2 & - & 0.2x_3 & = & 7.85 \\ 0.1x_1 & + & 7x_2 & - & 0.3x_3 & = & -19.3 \\ 0.3x_1 & - & 0.2x_2 & + & 10x_3 & = & 71.4 \end{array}$$

la cui soluzione esatta è $x_1 = 3$, $x_2 = -2.5$, $x_3 = 7$ e applichiamo il metodo di Gauss-Seidel. Per prima cosa, risolviamo ogni equazione rispetto all'incognita che si trova sulla diagonale principale.

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \tag{5.42}$$

$$x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \tag{5.43}$$

$$x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10} \tag{5.44}$$

Assumendo che x_2 e x_3 siano zero, usiamo la (5.42) per calcolare

$$x_1 = \frac{7.85}{3} = 2.616666667 \, .$$

Questo valore, insieme alla stima di $x_3 = 0$, viene sostituito nella (5.43) per calcolare

$$x_2 = \frac{-19.3 - 0.1(2.616666667) + 0}{7} = -2.794523810 \, .$$

La prima iterazione viene completata sostituendo i valori di x_1 e x_2 nella (5.44) per ottenere

$$x_3 = \frac{71.4 - 0.3(2.616666667) + 0.2(-2.794523810)}{10} = 7.005609524$$

Nella seconda iterazione vengono ripeture le stesse operazioni che danno come risultato

$$\begin{aligned} x_1 &= \frac{7.85 + 0.1(-2.794523810) + 0.2(7.005609524)}{3} = 2.990556508 \\ x_2 &= \frac{-19.3 - 0.1(2.990556508) + 0.3(7.005609524)}{7} = -2.499624684 \\ x_3 &= \frac{71.4 - 0.3(0.2990556508) + 0.2(-2.499624684)}{10} = 7.00029081 \ . \end{aligned}$$

Come si può vedere, il metodo converge alla soluzione esatta e ulteriori iterazioni migliorerebbero la precisione dei risultati. Già alla seconda iterazione la stima dell'errore associato alla soluzione è comunque abbastanza piccola:

$$\begin{aligned} \left| \frac{2.990556508 - 2.616666667}{2.990556508} \right| &= 0.125 \\ \left| \frac{-2.499624684 - (-2.794523810)}{-2.499624684} \right| &= 0.118 \\ \left| \frac{7.000290811 - 7.005609524}{7.000290811} \right| &= 0.00076 \ . \end{aligned}$$

La differenza tra i metodi di Gauss-Seidel e di Jacobi viene mostrata in Fig. 5.8. Nonostante il metodo di Jacobi converga più velocemente in alcuni casi particolari, solitamente l'uso delle migliori stime disponibili rende preferibile il metodo di Gauss-Seidel. Si può osservare che il metodo di Jacobi realizza un disaccoppiamento *simultaneo* delle equazioni e risulta pertanto attuabile in parallelo, mentre il metodo di Gauss-Seidel realizza un disaccoppiamento *in successione* ed è pertanto attuabile solo in modo sequenziale. Conseguentemente, con il metodo di Gauss-Seidel, il numero di iterazioni necessarie per ottenere una soluzione soddisfacente, dipende dall'ordine con cui le equazioni vengono risolte, cosa che non accade nel metodo di Jacobi.

5.6.5 Criterio di convergenza

Per quanto riguarda i metodi presentati, l'insieme delle soluzioni approssimate forma una successione convergente (sotto certe ipotesi) alla soluzione esatta. In generale, si può dimostrare che entrambi i metodi convergono alla soluzione esatta indipendentemente dalla soluzione di tentativo iniziale $\mathbf{x}^{(0)}$ se gli autovalori della matrice di iterazione hanno modulo < 1. Un criterio sufficiente per garantire la convergenza del metodo di Gauss-Seidel richiede che i coefficienti sulla diagonale principale debbano essere maggiori in valore assoluto della

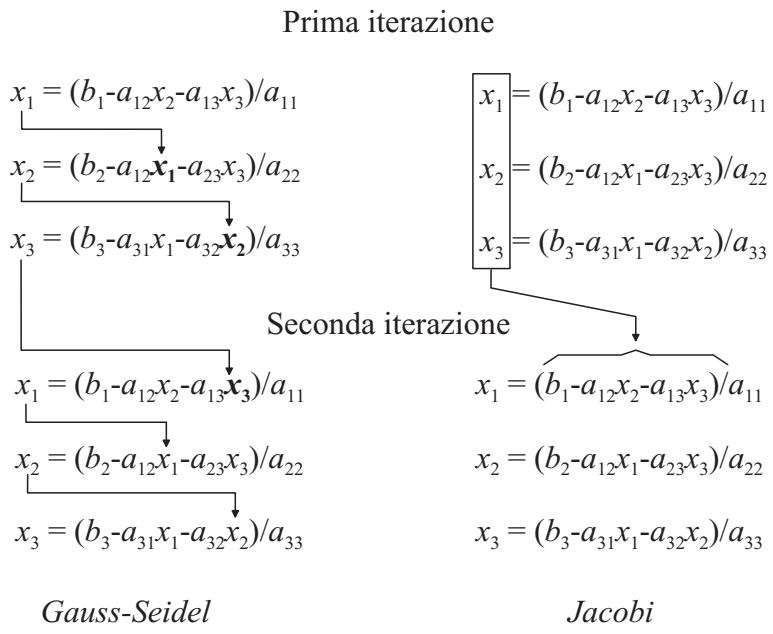


Figura 5.8: Differenza tra il metodo di Gauss-Seidel e il metodo di Jacobi nella risoluzione di sistemi di equazioni lineari algebriche

somma dei valori assoluti degli altri coefficienti della stessa riga o della stessa colonna, cioè:

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| , \qquad i = 1, \ldots, n$$

ovvero (5.45)

$$|a_{jj}| \geq \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}| , \qquad j = 1, \ldots, n$$

con disuguaglianza stretta valida almeno per un valore degli indici i (per la prima) o j (per la seconda). La (5.45) rappresenta un criterio sufficiente ma non necessario: in altre parole, il metodo può talvolta convergere anche se la (5.45) non viene soddisfatta, ma se la (5.45) è soddisfatta la convergenza è garantita. I sistemi di equazioni per i quali vale la (5.45) vengono detti *diagonalmente dominanti*; fortunatamente, molti problemi d'ingegneria di una certa importanza pratica hanno questa proprietà.

La convergenza di un metodo iterativo può essere verificata utilizzando il criterio:

$$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \|\mathbf{x}^{(k)}\|\varepsilon ,$$

dove k e $k-1$ sono, rispettivamente, l'iterazione attuale e quella precedente. Osserviamo infine che, mentre i metodi diretti in assenza di errori nella rappresentazione dei numeri forniscono la soluzione esatta del sistema, indipendentemente da quella che potrebbe essere la precisione richiesta dall'utente, i metodi iterativi, se convergenti, consentono invece

di arrestare il processo iterativo non appena la precisione desiderata è stata raggiunta. Va tuttavia sottolineato che un metodo iterativo risulterà efficiente solo se consentirà di raggiungere la precisione desiderata con un numero accettabile di iterazioni, ovvero di operazioni aritmetiche. Il costo computazionale dei metodi di Jacobi e di Gauss-Seidel è dell'ordine di n. Tuttavia, poiché per ogni iterazione la risoluzione del sistema disaccoppiato dà solo una soluzione approssimata, i metodi iterativi sono realmente vantaggiosi rispetto ai metodi diretti (quali l'eliminazione gaussiana o la decomposizione LU) quando il numero di iterazioni necessarie alla convergenza è dell'ordine di $n^{1/10}$.

La convergenza del metodo di Gauss-Seidel non implica quella di Jacobi, e viceversa. Tuttavia, quando entrambi convergono, la velocità di convergenza di Gauss-seidel è generalmente superiore. Nei due esempi che seguono mettiamo a confronto le prestazioni dei metodi di Jacobi e di Gauss-Seidel.

Esempio 1

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 2 & 6 & 2 \\ 1 & 2 & 4 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 3 \\ 10 \\ 7 \end{bmatrix}$$

In questo caso, come mostrano le tabelle seguenti, sia il metodo di Jacobi sia quello di Gauss-Seidel convergono alla soluzione esatta $\mathbf{x} = [1, 1, 1]^T$, ma il secondo converge più rapidamente.

- Jacobi -

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	1.666667	1.750000
2	1.027778	0.750000	0.666667
3	0.972222	1.101852	1.118056
4	1.005401	0.969907	0.956019
5	0.995370	1.012860	1.013696
6	1.000279	0.996978	0.994727
7	0.999250	1.001665	1.001441
8	0.999926	0.999770	0.999355
9	0.999862	1.000240	1.000134
10	0.999965	1.000001	0.999915
11	0.999971	1.000040	1.000008
12	0.999989	1.000007	0.999987
13	0.999993	1.000008	0.999999
14	0.999997	1.000002	0.999998
15	0.999998	1.000002	0.999999
16	0.999999	1.000001	1.000000
17	1.000000	1.000000	1.000000

- Gauss-Seidel -

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	1.333333	0.833333
2	0.833333	1.111111	0.986111
3	0.958333	1.018519	1.001157
4	0.994213	1.001543	1.000675
5	0.999711	0.999871	1.000137
6	1.000088	0.999925	1.000015
7	1.000030	0.999985	1.000000
8	1.000005	0.999998	1.000000
9	1.000000	1.000000	1.000000
10	1.000000	1.000000	1.000000

Esempio 2

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

La soluzione del sistema $A\mathbf{x} = \mathbf{b}$ è $\mathbf{x} = [1, 1, 1, 1]^T$. Il metodo di Jacobi non converge, mentre quello di Gauss-Seidel, la cui convergenza è assicurata dalle proprietà di A, produce le seguenti approssimazioni:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$x_4^{(k)}$
1	4.600000	-0.020000	0.556000	0.313600
2	3.647200	-0.017360	0.843328	0.529557
3	3.082754	-0.003280	0.976370	0.682186
4	2.750761	0.015841	1.022904	0.792918
5	2.557421	0.036440	1.022769	0.875289
6	2.446372	0.056622	0.999119	0.937971
7	2.383815	0.075454	0.965174	0.986618
8	2.349537	0.092552	0.928279	1.024993
9	2.331498	0.107832	0.892341	1.055661
10	2.322564	0.121369	0.859271	1.080416
11	2.318542	0.133313	0.829850	1.100545
12	2.316997	0.143841	0.804239	1.116998
13	2.316538	0.153133	0.782272	1.130492
14	2.316394	0.161362	0.763631	1.141582
15	2.316155	0.168680	0.747940	1.150701
16	2.315620	0.175224	0.734818	1.158197
17	2.314708	0.181112	0.723908	1.164350
18	2.313403	0.186446	0.714886	1.169389

```
  19        2.311724  0.191312  0.707466  1.173500
  20        2.309703  0.195785  0.701400  1.176839
   .
   .
   .
 100        2.030160  0.380834  0.738440  1.153740
   .
   .
   .
 400        1.405573  0.756235  0.897024  1.060527
   .
   .
   .
1000        1.062863  0.962217  0.984039  1.009382
```

5.6.6 Programma per i metodi di Jacobi e di Gauss-Seidel

Dall'applicazione delle formule iterative (5.39) e (5.41) si possono scrivere le procedure che implementano i metodi di Jacobi e di Gauss-Seidel, di cui proponiamo un esempio.

Metodo di Jacobi

```
#define n 50
#define EPS 1e-6                                  /* tolleranza prefissata */
#define MAXITER 100000                          /* numero max di iterazioni */

int i,j,k;                                        /* variabili di conteggio */
double A[n][n+1], Xold[n], Xnew[n];
double sum;                                        /* variabile temporanea */
int flag;          /* variabile di controllo per il criterio di arresto */

for(i=0;i<n;i++)                      /* inizializzazione della soluzione */
 Xold[i]=0;

flag=0;
k=0;

while((k<MAXITER) && (flag==0))                       /* ciclo principale */
 {
  k++;
  flag=1;
  for(i=0;i<n;i++)                 /* ciclo sul vettore delle incognite */
   {
    sum=0;
    for(j=0;j<n;j++)
     if(j!=i) sum+=A[i][j]*Xold[j];
    Xnew[i]=(A[i][n]-sum)/A[i][i];      /* calcolo nuova approssimazione */
    if (fabs(Xnew[i]-Xold[i]) > fabs(Xnew[i])*EPS) flag=0;
```

```
   }
  for(i=0;i<n;i++)                                    /* aggiornamento */
   Xold[i]=Xnew[i];
 }
```

La procedura opera su una matrice di n righe e $n+1$ colonne che memorizza la matrice costruita aggiungendo ad A il vettore dei termini noti $\mathbf{b}$, e sui vettori `Xnew` e `Xold` che contengono, rispettivamente, le soluzioni di tentativo all'iterazione corrente e a quella precedente. Il procedimento ha termine se la relazione `fabs(Xnew[i]-Xold[i]) > fabs(Xnew[i])*EPS` risulta *vera per tutti i valori di* `i`, oppure se si è raggiunto il numero massimo prefissato di iterazioni `MAXITER`. A tale scopo la variabile `flag` è posta al valore `1` prima del calcolo di ciascuna `Xnew[i]`, e viene posta al valore `0` se la relazione non è verificata per un qualsiasi valore di `i`.

Metodo di Gauss-Seidel

Nel metodo di Gauss-Seidel vengono utilizzati, nella valutazione della nuova x_i, i valori aggiornati di $x_1, \ldots, x_{i-1}$. Pertanto, la procedura può essere modificata aggiornando il valore di `Xnew[i]` direttamente nel ciclo di risoluzione sugli elementi del vettore delle incognite. Si può inoltre osservare che utilizzando direttamente la nuova soluzione di tentativo, non è necessario memorizzare tutti gli elementi del vettore soluzioni all'iterazione precedente (`Xold[i]`): è sufficiente una variabile `X` in cui viene memorizzato il nuovo valore approssimato della `i`-esima incognita.

```
#define n 50
#define EPS 1e-6
#define MAXITER 100000

int i,j,k;
double A[n][n+1], X, Xnew[n];
double sum;
int flag;

for(i=0;i<n;i++)                        /* inizializzazione della soluzione */
 Xnew[i]=0;

flag=0;
k=0;

while((k<MAXITER) && (flag==0))                        /* ciclo principale */
 {
  k++;
  flag=1;
  for(i=0;i<n;i++)                    /* ciclo sul vettore delle incognite */
```

```
   {
    sum=0;
    for(j=0;j<n;j++)
     if(j!=i) sum+=A[i][j]*Xnew[j];
    X=(A[i][n]-sum)/A[i][i];            /* calcolo nuova approssimazione */
    if (fabs(Xnew[i]-Xold[i]) > fabs(Xnew[i])*EPS) flag=0;
    Xnew[i]=X;                                          /* aggiornamento */
   }
 }
```

5.6.7 Miglioramento della convergenza con strategie di rilassamento

Il *rilassamento* introduce una leggera variazione nei metodi iterativi presentati allo scopo di favorirne la convergenza. Ogni nuovo valore delle $\mathbf{x}$, calcolato per mezzo della (5.39) o della (5.41), viene modificato secondo una media pesata dei risultati della iterazione corrente e della precedente:

$$x_i^{(k+1)} = \lambda x_i^{(k+1)} + (1 - \lambda) x_i^{(k)} \tag{5.46}$$

dove λ è un coefficiente di peso il cui valore è compreso tra 0 e 2. Se $\lambda = 1$, $(1 - \lambda)$ è uguale a zero e il risultato non viene modificato. Se, invece, λ ha un valore compreso tra 0 e 1, il risultato è effettivamente una media pesata tra il risultato dell'iterazione attuale e quello della precedente; in questo caso si parla di *sottorilassamento* e questa opzione viene utilizzata tipicamente per rendere convergente un sistema che non lo è. Per valori di λ tra 1 e 2, il valore ottenuto dall'iterazione attuale viene incrementato (in valore assoluto). In questo caso si assume implicitamente che la soluzione si stia muovendo nella direzione giusta, ma troppo lentamente; il maggior peso che viene assegnato alla soluzione per mezzo di λ ha quindi lo scopo di spingere più velocemente la soluzione stessa verso il risultato esatto. Questa opzione, detta *sovrarilassamento*, consente di accelerare la convergenza di un sistema che convergerebbe in ogni caso.

La scelta del valore più appropriato di λ dipende strettamente dal problema e viene spesso determinato per tentativi. Se un dato sistema di equazioni deve essere risolto una sola volta, la laboriosità richiesta dalla determinazione di λ non ha modo di essere ripagata; se, invece, il sistema deve essere risolto più volte, l'aumento di efficienza dovuto ad un'opportuna scelta di λ può essere sensibile.

5.7 Osservazioni conclusive

5.7.1 Confronto tra i metodi presentati in relazione alle caratteristiche della matrice dei coefficienti

Come abbiamo visto nei paragrafi precedenti, i metodi numerici per la risoluzione di sistemi lineari vengono suddivisi in due classi: metodi diretti e metodi iterativi. Per sistemi $A\mathbf{x} = \mathbf{b}$ con matrici A *dense*, cioè con la maggior parte degli elementi a_{ij} non nulli, i metodi diretti sono di solito i più efficienti. Tuttavia, in numerose applicazioni siamo chiamati a risolvere sistemi a matrice sparsa in cui molti elementi a_{ij} sono nulli. Ad esempio, nella risoluzione numerica di equazioni alle derivate parziali con metodi alle differenze finite, gli elementi non nulli costituiscono successioni regolari di pochi numeri distinti e generalmente disposti in configurazioni (o trame) regolari a banda. Quando i metodi di discretizzazione conducono alla risoluzione di sistemi lineari con matrice a banda, a blocchi o sparsa, lo sfruttamento della struttura della matrice consente di ridurre drasticamente i costi computazionali comportati dalle fattorizzazioni e dagli algoritmi di sostituzione in avanti e all'indietro. Nei metodi dell'eliminazione gaussiana o della fattorizzazione LU il numero di operazioni (moltiplicazioni) cresce con il cubo della dimensione del sistema ($O(n^3)$). Ciò comporta un elevato costo computazionale quando il sistema di equazioni da risolvere è di grosse dimensioni. Sfruttando la sparsità delle matrici, si può fare in modo di evitare la moltiplicazione se il coefficiente è nullo. Con questo accorgimento si può ricavare empiricamente che il costo del metodo diretto scende a $O(n^{1.5})$. Tuttavia, a causa dell'elevato ordine delle matrici risultanti dai metodi di discretizzazione, i metodi diretti non sempre sono utilizzabili. Infatti questi ultimi ottengono la soluzione $\mathbf{x}$ in un numero finito di passi mediante una successione (finita) di trasformazioni del problema iniziale in problemi equivalenti, cioè con la stessa soluzione $\mathbf{x}$, ma con matrici dei coefficienti diverse. Con il procedere del metodo di risoluzione il numero di elementi non nulli presenti in queste matrici generalmente cresce, dando luogo ad un fenomeno di *riempimento* o *fill-in* che può ben presto saturare lo spazio disponibile nella memoria centrale del calcolatore.

Per contrastare l'effetto di riempimento (nel corso dell'algoritmo di soluzione) si può sfruttare l'osservazione che nel metodo di eliminazione gaussiana l'ordine con cui si eliminano le incognite influenza la sparsità. E' necessario introdurre un criterio di ordinamento per righe e colonne che controlli il *fill-in*. La scelta del pivot (che poi è la variabile da eliminare) deve tener conto degli effetti di riempimento. Per ogni elemento di ogni riga e di ogni colonna si può ad esempio analizzare quanti coefficienti vengono riempiti e possibilmente si dovrebbero analizzare tutti i casi per ogni passo k tenendo conto anche di quello che succederebbe nei passi successivi. In questi termini, tuttavia il problema ha un'impostazione troppo complicata e porterebbe ad una esplosione combinatoria del problema. Allora ci si accontenta di esaminare gli effetti soltanto rispetto al passo di sostituzione immediatamente successivo. Un secondo aspetto riguarda il fatto che la scelta dovrebbe tener conto anche dei criteri di precisione di calcolo (per garantire l'accuratezza della soluzione). Nonostante questi accorgimenti, in questi casi è preferibile, e spesso indispensabile, utilizzare metodi iterativi, i quali, operando sempre e solo con gli elementi della matrice iniziale A,

generano una successione infinita di vettori convergente, sotto opportune condizioni, alla soluzione cercata. Poiché il processo iterativo lascia inalterata la matrice A, è sufficiente memorizzare gli elementi non nulli di A.

5.7.2 Strutture dati per matrici sparse

Il problema è quello duplice di (1) ridurre lo spazio di memoria e (2) rendere semplici le implementazioni degli algoritmi (ad esempio gli scambi di righe e colonne). A tale scopo può essere conveniente utilizzare strutture dati dinamiche come ad esempio grafi o liste bidimensionali. La trattazione di questo tipo di strutture va al di là dello scopo di queste note e si rimanda a testi specialistici e/o a insegnamenti specifici del Corso di Laurea. In generale, le liste permettono un miglior utilizzo della memoria fisica del calcolatore in modo che meglio corrisponda alla struttura della memoria astratta, data dal numero di elementi della matrice. L'idea base consiste nel concatenare tra loro, mediante puntatori, diversi elementi di una lista costituiti di più parti quali ad esempio il valore del coefficiente della matrice e la relativa coppia di indici che ne identifica la posizione nelle righe e nelle colonne. Ad esempio, considerando una struttura del tipo:

valore	indice riga	indice colonna	puntatore riga succ.	puntatore colonna succ.

la matrice

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 9 & 4 & 6 & 0 \\ 0 & 8 & 7 & 5 \\ 0 & 0 & -1 & 3 \end{bmatrix}$$

può essere memorizzata, utilizzando la lista dinamica rappresentata in Fig. 5.9.

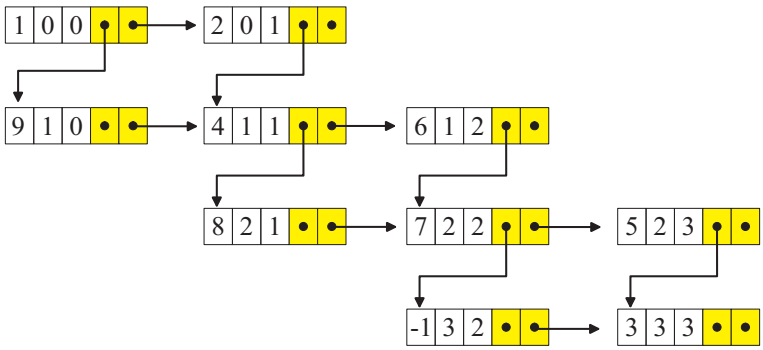


Figura 5.9: Schematizzazione grafica della memorizzazione della matrice dei coefficienti mediante una lista dinamica bidimensionale. Il vantaggio di tale rappresentazione risulta evidente se la matrice è molto sparsa

Bibliografia

S.C. Chapra, R.P. Canale, *Metodi numerici per l'ingegneria*, McGraw-Hill Italia, 1988

A. Quarteroni, R. Sacco, F. Saleri, *Matematica Numerica*, Springer-Verlag Italia, 1998

G. Monegato, *Fondamenti di calcolo numerico*, Libreria Editrice Universitaria Levrotto&Bella, Torino, 1990

B. Fadini, C. Savy, *Fondamenti di Informatica - fondamenti di programmazione*, Liguori editore, 1991

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

Newton A.R., Sangiovanni-Vincentelli A.L., *Relaxation-based electrical simulation.* IEEE Trans. on CAD, October 1984, p.308

Capitolo 6

Considerazioni finali

I metodi numerici possono essere definiti scientifici in quanto rappresentano tecniche sistematiche per la risoluzione di problemi matematici. La loro applicazione efficiente in campo ingegneristico, però, richiede anche un certo buon senso, l'abilità di trovare dei compromessi e una capacità di giudizio soggettivo: ogni particolare problema, infatti, può essere risolto per mezzo di diverse tecniche numeriche, eseguibili su un gran numero di calcolatori differenti. Pertanto la valutazione dell'eleganza ed efficienza di diversi approcci alla risoluzione di problemi è strettamente individuale ed è legata all'abilità di ciascuno nello scegliere saggiamente tra le diverse opzioni disponibili. Sfortunatamente, come accade per ogni processo intuitivo, è difficile apprendere quali siano i fattori che influenzano questa scelta: solo l'esperienza consente di comprendere e sfruttare queste nozioni. Dato, però, che queste nozioni giocano un ruolo così importante nell'applicazione pratica dei metodi numerici, presenteremo in questo capitolo alcuni dei compromessi che devono essere presi in considerazione nella scelta di un metodo numerico e degli strumenti per la sua implementazione.

Prendiamo dunque in esame sette differenti fattori, o compromessi, che devono essere presi in considerazione quando si sceglie un metodo numerico per la risoluzione di un particolare problema.

Tipo del problema matematico

In queste note abbiamo trattato diversi tipi di problemi matematici:

- Integrazione e derivazione di una funzione
- Ricerca delle radici di una equazione
- Ricerca di massimi e minimi relativi di funzioni di una variabile
- Sistemi di equazioni lineari algebriche

Probabilmente, l'importanza degli aspetti applicativi dei metodi numerici vi apparirà evidente mentre affronterete la soluzione di un problema relativo a una delle categorie appena menzionate: il problema in questione richiederà l'uso di tecniche numeriche in quanto l'approccio analitico non sarà in grado di dare efficientemente i risultati richiesti. È indubbio che, prima o poi, l'attività professionale di un ingegnere richieda la soluzione di problemi classificabili in una delle categorie viste sopra; pertanto, lo studio dei metodi numerici e la scelta di strumenti per il calcolo automatico dovrebbe, come minimo, prendere in considerazione tutte queste classi fondamentali di problemi. Problemi più complessi possono richiedere la soluzione di sistemi di equazioni algebriche non-lineari, l'interpolazione con curve a più variabili, l'ottimizzazione di parametri, la programmazione lineare, i problemi agli autovalori e le equazioni differenziali alle derivate ordinarie o parziali: questi particolari problemi matematici richiedono la disponibilità di una maggiore potenza di calcolo e l'applicazione di tecniche numeriche più avanzate, che non sono state descritte in queste note. In tali casi la soluzione può essere ricercata in testi più avanzati come Press *et al.*, 1992.

Tipo, disponibilità, precisione, costo e velocità dei calcolatori

Gli strumenti di calcolo a vostra disposizione potrebbero appartenere a categorie diverse: dalla calcolatrice tascabile al mainframe. Il metro di valutazione, però, non si basa esclusivamente sulla potenza di calcolo teoricamente disponibile, ma è una funzione complessa di costo, semplicità d'uso, velocità, affidabilità, ripetibilità e precisione. Il recente rapido sviluppo delle prestazioni dei personal computer ha fatto sensibilmente sentire la sua influenza sulla professione dell'ingegnere. È dunque facile aspettarsi che questa rivoluzione si diffonda sempre più, man mano che la tecnologia avanza, perché il personal computer offre un eccellente compromesso tra semplicità d'uso, costo, precisione, velocità e capacità di memoria; inoltre, il personal computer si adatta facilmente alla soluzione dei più comuni problemi in campo tecnico. Per questi motivi, i metodi numerici illustrati in queste note sono stati scelti in modo che fossero implementabili su calcolatori di questa classe.

Costi di sviluppo, di acquisto e di esecuzione dei programmi

Una volta identificato il tipo di problema da risolvere e dopo aver scelto lo strumento di calcolo adeguato, è il momento di prendere in considerazione i costi dei programmi e quelli associati alla loro esecuzione: lo sviluppo dei programmi, infatti, può costituire una parte notevole dello sforzo richiesto da un tipico problema in campo tecnico e può, pertanto, richiedere una spesa significativa. A questo proposito, è bene sottolineare l'importanza di una conoscenza approfondita degli aspetti teorici e pratici relativi ai metodi numerici

che si intendono utilizzare. I programmi disponibili in commercio solitamente sono in grado di risolvere solo alcuni dei problemi tipici dell'ingegneria e il loro costo può essere elevato; inoltre, questi programmi devono essere utilizzati con prudenza da chi non ha un'approfondita conoscenza della logica che sta alla loro base (caso tipico della maggioranza degli utenti). L'alternativa è data da programmi di utilità generale, come MATLAB, che implementano metodi numerici facilmente adattabili a una grande quantità di problemi diversi. I costi di sviluppo dei programmi si ripagano in fase di esecuzione dei programmi stessi se questi sono stati scritti tenendo conto delle tecniche appropriate e se sono stati provati a fondo.

Caratteristiche dei metodi numerici

Quando i costi del calcolatore e dei programmi sono alti, o se la disponibilità del calcolatore è limitata, vale al pena di scegliere accuratamente il metodo numerico da utilizzare in modo che si adatti perfettamente alla situazione. Al contrario, se la soluzione del problema è ancora in una fase esplorativa e l'accesso al calcolatore e il suo costo non pongono difficoltà, la scelta migliore può essere quella di un metodo numerico che funzioni sempre anche se non è necessariamente il più efficiente dal punto di vista del calcolo. Qualunque metodo numerico disponibile richiede una decisione per quanto riguarda i compromessi appena illustrati più i seguenti altri:

a. *Quantità dei dati iniziali (o di tentativo).* Alcuni dei metodi numerici per la ricerca delle radici di un'equazione o per la risoluzione di equazioni differenziali richiedono che l'utente specifichi dei valori iniziali. I metodi semplici richiedono un solo valore, mentre metodi più sofisticati possono richiederne di più. Il compromesso che si presenta è il seguente: i vantaggi di complessi metodi avanzati possono essere annullati dalla necessità di specificare diversi valori iniziali. La decisione per ogni particolare problema dovrà essere guidata dall'esperienza e dalla capacità di giudizio dell'utente.

b. *Velocità di convergenza.* Certi metodi numerici convergono più rapidamente di altri; per ottenere questa maggiore velocità, però, può essere necessaria la determinazione di molti valori iniziali e una programmazione più elaborata rispetto a un metodo che presenta una convergenza meno rapida. Anche in questo caso, è il giudizio dell'utente che deve guidare la scelta: non sempre "più veloce" significa migliore.

c. *Stabilità.* Alcuni metodi numerici per la ricerca delle radici delle equazioni o per la soluzione di sistemi di equazioni lineari in certi casi possono divergere anziché convergere alla soluzione corretta. Viene da domandarsi per quale motivo tali metodi ci possano interessare: la risposta è che, quando funzionano, possono essere molto efficienti. Anche per quanto riguarda questo aspetto, quindi, ci si trova di fronte a un compromesso: l'utente deve decidere se il problema da risolvere giustifica gli sforzi richiesti per applicare un metodo che non sempre converge.

d. *Accuratezza e precisione.* Alcuni metodi numerici sono per loro natura più accurati e precisi di altri; un tipico esempio è dato dai vari metodi disponibili per l'integrazione numerica. Di solito, le prestazioni di metodi di scarsa accuratezza possono essere migliorate riducendo il passo o aumentando il numero di applicazioni in un dato intervallo. È più conveniente utilizzare un metodo poco accurato con un passo piccolo o un metodo molto accurato con un passo grande? Questo dilemma deve essere risolto caso per caso, considerando anche altri fattori come costi e facilità di programmazione. Inoltre, l'utente deve anche tenere conto degli errori di arrotondamento quando metodi con bassa accuratezza, applicati più volte, fanno crescere sensibilmente la quantità di calcoli necessaria. La decisione per quanto riguarda questo fattore può essere influenzata decisamente dal numero di cifre significative adottate dal calcolatore.

e. *Campo di applicazione.* Alcuni metodi numerici possono essere applicati a una ristretta classe di problemi o a problemi che soddisfano particolari condizioni matematiche; altri metodi, invece, non presentano tali limitazioni. È quindi necessario stabilire se vale veramente la pena di sviluppare un programma che si basa su tecniche valide solo per un numero limitato di problemi. Il fatto che tali tecniche siano ampiamente diffuse suggerisce che i vantaggi ad esse associati possono spesso pesare più degli svantaggi. Come sempre è una questione di compromessi.

f. *Particolari condizioni per l'applicazione.* Alcune tecniche numeriche tentano di migliorare l'accuratezza e la velocità di convergenza utilizzando dati particolari: per esempio, l'accuratezza può essere migliorata se si conosce il valore stimato o teorico dell'errore. Questi miglioramenti, però, richiedono in generale un incremento dei costi associati all'uso del calcolatore o un aumento della complessità del programma.

g. *Lavoro di programmazione richiesto.* Gli sforzi per migliorare la velocità di convergenza, la stabilità e l'accuratezza permettono, in genere, di dare libero sfogo alla creatività e all'ingegnosità. Se gli aumenti di prestazioni possono essere ottenuti senza aumentare la complessità del programma, le soluzioni trovate possono essere considerate eleganti e possono trovare un uso immediato nella professione ingegneristica. Se, però, il programma risultante è molto più complesso della versione base, la situazione che si presenta richiede ancora la valutazione di un compromesso e il nuovo metodo potrebbe risultare sfavorito.

Riassumendo la discussione appena fatta sui criteri di scelta dei metodi numerici possiamo dire che, in pratica, i due parametri fondamentali su cui si basa la decisione sono il costo e l'accuratezza, dove i costi sono quelli relativi al tempo di calcolo e allo sviluppo dei programmi, mentre la scelta del livello appropriato di accuratezza dipende dall'etica professionale e dalla capacità di giudizio di chi sviluppa i programmi.

Proprietà matematiche di funzioni, equazioni e dati

Nella scelta di un particolare metodo numerico, di un tipo di calcolatore e dei programmi appropriati, bisogna tenere conto della complessità della funzione, dell'equazione e dei dati che costituiscono la formulazione matematica del problema. Equazioni semplici e insiemi di dati regolari possono essere trattati adeguatamente con algoritmi numerici semplici e calcolatori di basso costo; al contrario, equazioni complesse e dati molto dispersi richiedono algoritmi e strumenti di calcolo più potenti.

Semplicità d'uso (buona "interfaccia" con l'utente)

Alcuni metodi numerici sono semplici da usare, altri meno; questa caratteristica può far pendere la bilancia a favore di un metodo confrontato con un altro equivalente dal punto di vista dei risultati. La stessa considerazione si applica anche alla scelta tra programmi sviluppati autonomamente e programmi disponibili in commercio: un programma costoso ma semplice da usare può essere più conveniente di un programma a buon mercato ma scomodo; i costi aggiuntivi dovuti alle rielaborazioni necessarie per rendere utilizzabile un programma scomodo possono annullare il risparmio sul prezzo di acquisto o sui costi di sviluppo.

Manutenzione

I programmi per la risoluzione di problemi di ingegneria richiedono manutenzione in quanto, durante l'uso, inevitabilmente si manifestano errori, difficoltà, necessità di cambiamenti o di trasporto su calcolatori differenti. Si rende così necessaria la "manutenzione" dei programmi che può consistere nello sviluppo di ulteriore codice o nell'espansione della documentazione. Ovviamente, programmi e algoritmi numerici semplici sono più facili da mantenere.

Bibliografia

S.C. Chapra, R.P. Canale, *Metodi numerici per l'ingegneria*, McGraw-Hill Italia, 1988

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes in C*, Cambridge University Press (2^a ed.), 1992

Appendice A

Norme di vettore

Una norma di vettore su R^n è una funzione che ad ogni vettore $\mathbf{x} \in R^n$, $\mathbf{x} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$, associa un numero reale, che indichiamo con $\|\mathbf{x}\|$, con le seguenti proprietà:

(i) $\|\mathbf{x}\| > 0$ per ogni $\mathbf{x} \neq \mathbf{0}$ e $\|\mathbf{x}\| = 0$ se e solo se $\mathbf{x} = \mathbf{0}$,

(ii) $\|c\mathbf{x}\| = |c|\|\mathbf{x}\|$ qualunque sia $c \in R$,

(iii) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

Le relazioni (ii) e (iii) ci permettono di dedurre che per ogni norma abbiamo

$$\|\mathbf{x} - \mathbf{y}\| \geq |\|\mathbf{x}\| - \|\mathbf{y}\|| \ , \qquad \forall \mathbf{x}, \mathbf{y} \in R^n \ .$$

Le norme più frequentemente usate sono le seguenti:

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| \qquad \text{norma infinito} \tag{6.1}$$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \tag{6.2}$$

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} = \sqrt{\mathbf{x}^T \mathbf{x}} \qquad \text{norma euclidea.} \tag{6.3}$$

Esse sono casi particolari della più generale *norma p*, definita dalla relazione

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \ , \qquad p \geq 1 \ .$$

Infatti la (6.1) corrisponde al caso limite $p = \infty$, la (6.2) a $p = 1$ e la (6.3) a $p = 2$.

Parte II

Introduzione alla programmazione integrata MATLAB

ATTUALE TENDENZA DEL SOFTWARE

- Semplificare e velocizzare lo sviluppo delle applicazioni

- Utilizzo di paradigmi di *programmazione visuale* (ambienti RAD, *Rapid Application Development*):
  - linguaggi tradizionali
    - Basic, C/ C++
  - nuovi linguaggi
    - gestione di strumentazione (LabVIEW)
    - gestione web (Java)

ATTUALE TENDENZA DEL SOFTWARE

- Utilizzo di paradigmi di *programmazione classica* con apporto di *funzioni specifiche* precompilate

  - linguaggi tradizionali
    - C, C++ (NAG library)

  - nuovi *ambienti di sviluppo* (con interfaccia utente interattiva di tipo **prompt**)
    - calcolo simbolico (Mathematica)
    - calcolo numerico (MATLAB)

CALCOLO SIMBOLICO VS. CALCOLO NUMERICO

- Mathematica

```
In[1]:=Expand[(a+b)^2]
Out[1]=a² + 2ab + b²
```

se a=1/2 e b=1/3

```
In[4]:=Expand[(a+b)^2]

Out[4]= 25/36
```

- MATLAB

```
» (a+b)^2
??? Undefined function or variable 'a'.
```

se a=1/2 e b=1/3

```
» (a+b)^2

ans = 0.6944
```


ATTUALE TENDENZA DEL SOFTWARE

- Vantaggi
  - apprendimento veloce
  - semplice anche per non esperti
  - algoritmi trasparenti
  - pensare al proprio problema
  - ambienti completi :
    - editing, compilazione e debugging
    - gestione codice e gestione dati

- Svantaggi
  - minore flessibilità
  - codice non ottimizzato
  - tendenza ad una eccessiva semplificazione

MATLAB

- Nel corso degli anni MATLAB ha beneficiato del feedback degli utenti.
- Il programma è largamente impiegato sia in *campo educativo* sia in *campo applicativo*:
  - nelle *università* è utilizzato sia come strumento per la ricerca sia come strumento di apprendimento e di esercitazione con il quale implementare le nozioni apprese durante i corsi teorici
  - nell'*industria* è lo strumento preferito per la realizzazione di progetti di ricerca, sviluppo e analisi.

MATLAB

- È utilizzato in tutti i tipi di calcolo perché
  - si impara ed usa velocemente
  - è un linguaggio ad alto livello per le operazioni matriciali
  - incoraggia a trovare *soluzioni vettoriali*
  - può essere utilizzato interattivamente
  - fornisce molte *funzioni grafiche*
  - può essere interfacciato con C/C++ e Fortran
  - presenta diversi tipi di interfaccia per le diverse aree di applicazione
  - nasconde all'utente i *dettagli architetturali o algoritmici* non necessari, oppure ne permette la modifica

PSE

- Queste caratteristiche rendono MATLAB
  - un PSE (*Problem Solving environment*): sistemi software che forniscono tutti gli strumenti per risolvere problemi in una particolare area (e.g. Microsoft Word).
  - un "*Rapid Prototyping Environment*": sistemi software che permettono di testare algoritmi ed idee rapidamente e facilmente.

TESTI DI RIFERIMENTO

- Cavallo, Setola, Vasca, *Guida Operativa a MATLAB*, Liguori Editore, Napoli, 1994
- I testi forniti con MATLAB
  - I manuali sia cartacei che online (pdf)
  - L'help online disponibile in MATLAB
- Materiale disponibile in rete:
  - http://www.mathworks.com
  - ...

SOFTWARE DISPONIBILE

- Altri prodotti simili a MATLAB, anche per SO diversi (OSF, Linux) sono:
  - **gnuplot** (in qualsiasi distribuzione)
  - **Octave** (http://www.octave.org/)
  - **Scilab** **(**http://scilabsoft.inria.fr/)

MATLAB

- MATLAB (*MATrix LABoratory*) è un ambiente di sviluppo interattivo per il calcolo scientifico
- Originariamente sviluppato come interfaccia per il software matriciale LINPACK e EISPACK

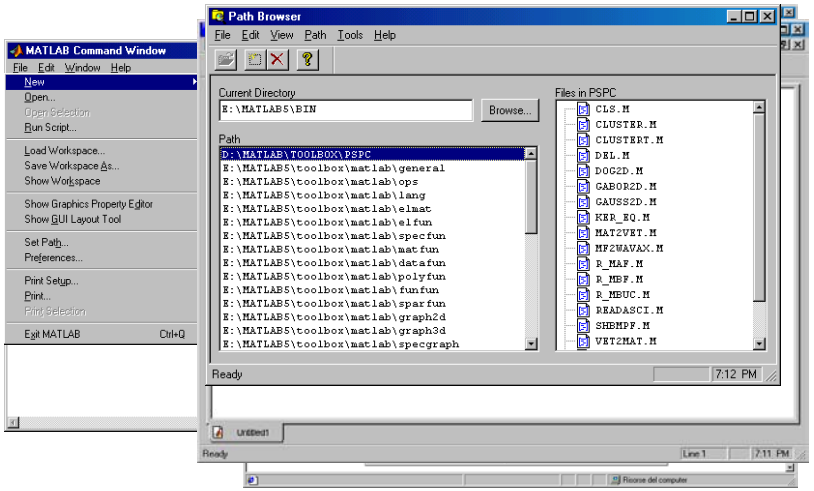
→ L'elemento base è la **matrice**, che non richiede dimensionamento

→ Ideale per risolvere problemi con formulazione matriciale o vettoriale

MATLAB

- MATLAB è composto di 6 parti principali:
  - L'ambiente di lavoro
  - Le librerie di funzioni matematiche
  - Il sistema grafico
  - Il linguaggio
  - API
  - Toolbox

AVVIO DI MATLAB



L'AMBIENTE MATLAB

- Sono forniti generici comandi di sistema per manipolare i file: ls, cd, delete, type …
- MATLAB esegue funzioni che sono nel path o nella directory corrente. Si aggiungono percorsi al path attraverso il menu File e Set Path
- Si possono eseguire programmi esterni tramite il carattere !

>>!notepad

Con il simbolo >> indico il prompt di MATLAB

L'AMBIENTE MATLAB

- Il comando help

>>help sqrt

Caratteri maiuscoli per evidenziare, ma tutte le funzioni devono essere chiamate in minuscolo.

SQRT Square root.

SQRT(X) is the square root of the elements of X. Complex results are produced if X is not positive.

See also SQRTM.

L'AMBIENTE MATLAB

- *help* fornisce una lista delle categorie delle funzioni
- *help categoria* mostra le funzioni di quella categoria
- *helpwin* genera una finestra di help
- *lookfor keyword* cerca le funzioni che corrispondono alla parola chiave fornita
- *helpdesk* fornisce la finestra di aiuto in html
- *doc funzione* richiama helpdesk per la data funzione

ALCUNI CARATTERI SPECIALI

- % è il commento
- … è la continuazione sulla riga successiva
- = è l'operatore di assegnamento
- == è l'operatore di uguaglianza
- ; impedisce l'echo su monitor
- , separa argomenti o comandi
- Ctrl-C termina l'esecuzione di un comando

L'AMBIENTE MATLAB

- Lo *workspace* è l'area di memoria accessibile dal prompt, dove si lavora.
- Per visualizzare i dati si utilizzano
  - *who* e *whos*
- Per cancellare dati in memoria
  - *clear nome_variabile*

```
>> a=[1 2 3];  ii=1;
>>who
Your variables are:
a       ii
>> whos
  Name      Size          Bytes  Class
  a         1x3              24  double array
  ii        1x1               8  double array
Grand total is 4 elements using 32 bytes
>>clear a,  whos
  Name      Size          Bytes  Class
  ii        1x1               8  double array
Grand total is 1 elements using 8 bytes
```


L'AMBIENTE MATLAB

- Per salvare l'intero workspace si usa *save nomefile*, per caricarlo si usa *load nomefile*
- Per salvare variabili in formato binario si usa *save nomefile variabili*, per caricarle *load nomefile*

```
>>a=[1 2];  b=[3;4];
>>save ab
>>ls ab
 ab.mat
>>clear a b
>>a
??? Undefined function or variable 'a'.
```

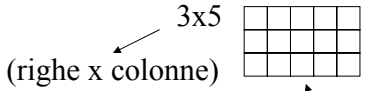
estensione .mat

```
>>load ab
>>a
 a = 1  2
>>b
 b=  1
     2
```

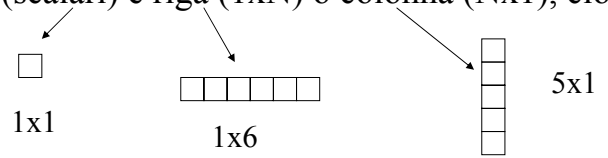

DEFINIZIONE DI VARIABILI

- MATLAB non usa *definizione di tipo* o *dichiarazione di dimensioni*
- Crea automaticamente la variabile digitata
- I nomi delle variabili sono *case sensitive*
- Devono iniziare con una *lettera* e possono contenere *31 caratteri* (lettere, numeri e underscore, ma non punti)

STRUTTURE DATI: MATRICI



- L'elemento base in MATLAB è la *matrice rettangolare*: particolare interesse per quella 1x1 (scalari) e riga (1xN) o colonna (Nx1), cioè vettori



Non c'è *dichiarazione* di variabili, non si usa la *definizione* di tipo e non è richiesto *dimensionamento*

CREARE MATRICI

- Si possono creare matrici in diversi modi:
  - Scrivere esplicitamente gli elementi
  - Caricare gli elementi da file esterni di dati
  - Generare gli elementi utilizzando le *built-in function*
  - Generare gli elementi utilizzando i propri **M-file** (*file che contengono il codice sorgente dei programmi MATLAB ed hanno estensione .m*)

CREARE MATRICI

- Per scrivere esplicitamente gli elementi:
  - Separare gli elementi di una riga con *spazi* o *virgole ,*
  - Finire le righe con *punto e virgola ;*
  - Tutti gli elementi devono essere racchiusi tra *parentesi quadrate* [ ]

```
>>A = [ 16 3 2 13 ; 5 10 11 8 ; 9 6 7 12 ; 4 15 14 1 ]
A = 16   3   2  13
     5  10  11   8
     9   6   7  12
     4  15  14   1
```


ELEMENTI DI UNA MATRICE

- Per accedere all'elemento nella *riga i* e *colonna j* si usa la notazione *A(i,j)*

```
>>A(2,3)
  ans=  11
```

- Se a=[1 2 ; 3 4] ;

N.B.
Gli indici devono essere interi positivi (non 0)

```
>> t=a(2,3)
      ???  Index exceeds matrix dimensions.
>>a(2,3)=5
     a = 1   2   0
         3   4   5
```

Possibili errori legati al non dimensionamento delle matrici

SCALARI E VETTORI

```
>>a=1
a =
   1
```

Scalare 1x1

```
>>b=[1  2  3  4  5]
b =
   1   2   3   4   5
```

vettore riga 1x5

```
>>b(3)
ans =
   3
```

```
>>c=[1; 2; 3]
c =
   1
   2
   3
```

vettore colonna 3x1

```
>> c(2)
ans =
   2
```

variabile predefinita

DIMENSIONI

- Per conoscere le dimensioni di una matrice si usa la funzione *size()*: *[r,c]=size(M)*

```
>>a=[1  2  3; 4  5  6];
>> [r,c] = size(a)
r  =  2
c  =  3
```

Notazione usata per gestire output multipli da una funzione

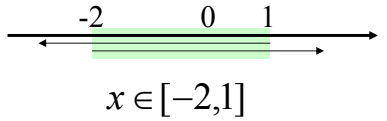
- Per conoscere il numero di elementi di un vettore si usa la funzione *length()*: *n=length(M)*

```
>>a=[1 2 3 4];  length(a)
ans  = 4
```


L'OPERATORE :

- Vediamo come rappresentare *insiemi numerici*: insiemi di numeri in cui l'informazione è contenuta nel valore iniziale, nel valore finale e nel passo di "campionamento" usato (il valore della differenza tra due numeri contigui dell'insieme considerato)
- Matematicamente si considera un *insieme infinito di valori* compresi tra due estremi:

$$\sqrt{(1-x)(2+x)}$$



L'OPERATORE :

- A causa della precisione finita dei numeri in un calcolatore, si considera un *insieme finito e quindi discreto* di valori compresi tra due estremi (e.g. [-2, 1] con incrementi di 0.1: quindi l'insieme di numeri {-2.0, -1.9, -1.8 … 0.8, 0.9, 1.0})

```
>>1:10
   ans = 1   2   3   4   5   6   7   8   9  10
>>10: -2 :5
    ans = 10   8   6
>>0:pi/4:pi
    ans = 0   0.7854   1.5708   2.3562   3.1416
```


L'OPERATORE :

- È di fondamentale importanza per creare *vettori*
- Per creare *tabelle di valori di funzioni* (dominio e condominio) si utilizza l'operatore :

Rappresentazione a precisione finita

```
>> x=0 : pi/4 : pi;
>> sin(x)
ans =  0   0.7071   1.0000   0.7071   0.0000
```

Formulazione vettoriale: equivalente a ciclo for e sin(x(i))

MATRICI

- 4 funzioni base
  - zeros
  - ones
  - rand
  - randn

```
>> a=2*ones(2,3)
     a =2    2    2
        2    2    2
>>b=rand(3,3)
   b = 0.9501   0.4860   0.4565
       0.2311   0.8913   0.0185
       0.6068   0.7621   0.8214
```

- Per eliminare righe o colonne utilizzare []

```
>>a=[1  2  3; 4  5  6]
    a =1    2    3
       4    5    6
>>a(1,:)=[]
    a=4    5    6

>>a=[1  2  3; 4  5  6];

>>a(:,1)=[]
     a = 2    3
         5    6
```

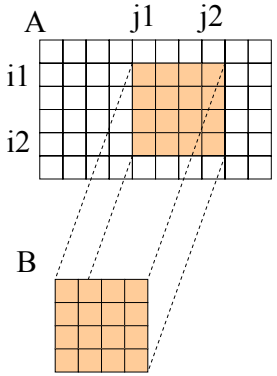

MATRICI

- Per gestire le matrici si fa riferimento all'algebra delle matrici. Si considerano solo l'estrazione di sottomatrici e le operazioni di addizione e sottrazione
- Si considera l'operazione elemento per elemento: funzioni (come quella vista: *sin(x)*) e operazioni in cui le variabili sono trattate come "scalari". Cioè la funzionalità viene applicata ai singoli elementi della matrice

MATRICI

- La gestione delle sottomatrici avviene per mezzo di indici vettoriali

```
>>a=[1 2 3 4; 5 6 7 8; 9 10 11 12]
a =   1    2    3    4
      5    6    7    8
      9   10   11   12
>>a(2:3,2:4)
ans =
    6    7    8
   10   11   12
```



B=A(i1:i2,j1:j2);

MATRICI

- Somma e sottrazione:

```
>>a=[1  2; 3  4];b=ones(2,2);
>>c=a+b
   c= 2   3
      4   5
>>c=a-3
   c= -2  -1
      0   1
>>a+ones(3,3)
??? Error using ==> +
Matrix dimensions must agree.
```

- Moltiplicazione

```
>>A=a*a
   A=   7   10
       15   22
>>(2*ones(1,2)) * a
     ans =  8   12
>>a*pi
    ans =3.1416    6.2832
         9.4248   12.5664
```

MATRICI

- Per operare elemento per elemento si pone il . prima del corrispondente operatore:

```
>>a=[1  2; 3  4]; a.*a
     ans=  1    4
           9   16
>>a./(2*ones(2,2))
      ans =0.5000    1.0000
           1.5000    2.0000
>> a.\(2*ones(2,2))
      ans =2.0000    1.0000
           0.6667    0.5000
```


ESPRESSIONI

- I blocchi costituenti le espressioni sono (ricordarsi che agiscono su intere matrici):

- Variabili *(appena viste)*
- Numeri
- Operatori
- Funzioni

NUMERI E OPERATORI

- Si usa la notazione decimale, e per la notazione scientifica e i o j per i numeri immaginari

3	-99
0.0120	-9.1233
1.34512 e 23	6.34567 e -20
-2.3 i	2.1 + 3.2 i

- Sono memorizzati in formato *long* IEEE

- + somma
- – sottrazione
- * moltiplicazione
- / divisione
- \ divisione sinistra
- ^ esponente
- ' trasposta (compl. con.)
- . operatore elemento per elemento
- ( ) ordine di valutazione

NOMI BULT-IN

- È utile non sovrascrivere i seguenti nomi built-in:
  - ans
  - pi (3.1415)
  - eps (2.2204e-016)
  - realmin (2.2251e-308)
  - realmax (1.7977e+308)
  - i, j ($\sqrt{-1}$)
  - nargin
  - nargout
  - Inf
  - NaN
  - flops

- computer
- date
- clock
- cputime

FUNZIONI

- Arrotondamento

```
>>round(2.3)
ans =
    2

>>floor(2.3)
ans =
    2

>>ceil(2.3)
ans =
    3
```

```
>>round(2.7)
ans =
    3

>>floor(2.7)
ans =
    2

>> ceil(2.7)
ans =
    3
```


FUNZIONI

- Approssimazioni razionali e fattorizzazione intera

```
>>rem(11,3)
ans =
    2

>> rats(1/2 -1/3 +1/5)
ans =
    11/30

>>gcd(27,72)
ans =
    9
```


FUNZIONI

- Aritmetica complessa

```
>>z1=1+3*j;z2=2-5*j;
>> z1*z2
ans =
  17.0000 + 1.0000i

>>real(z1)
ans =
    1

>> imag(z1)
ans =
    3
```

```
>> j=1;
>>z1=1+3*j
z1 =
    4

>> j=sqrt(-1)
j =
       0 + 1.0000i

>> z1=1+3*j
z1 =
   1.0000 + 3.0000i
```


FUNZIONI

- Esponenziali, logaritmiche, trigonometriche e specifiche di particolari ambiti scientifici.
- In generale possono operare su dati complessi e matriciali

```
>>pow2(10)
ans =
      1024

>>log10(10)
ans =
    1
```

```
>> asin(0.6)
ans =
0.6435

>>factorial(15)
ans =
  1.3077e+012
```

OPERATORI RELAZIONALI E LOGICI

- < minore
- <= minore o uguale
- > maggiore
- >= maggiore o uguale
- == uguale
- ~= diverso

- & and
- | or
- xor or esclusivo
- ~ not

- Il valore *falso* è indicato con 0
- Il valore *vero* con 1

```
>>a=[1 2 3; 2 2 5];
>>A=((a/2)==1)
A =
    0    1    0
    1    1    0
```


POLINOMI

- I *polinom*i si rappresentano come *vettori riga* contenenti i coefficienti in ordine di potenze decrescenti

  – $p(x)=x^3 + 3x^2 + 2x +10$ si rappresenta come p=[1 3 2 10]

  – $p(x)=x^3 + 1$ si rappresenta come p=[1 0 0 1]

POLINOMI

- La costruzione di un polinomio con specifiche radici si esegue con la funzione *poly()*. Radici: -2 -j3, -2 +j3, -5; ovvero il polinomio p(x)=(x+2+j3)(x+2-j3)(x+5)

```
>>r=[-2-j*3,-2+j*3,-5];
>>p=poly(r)
p =
    1    9   33   65
```


POLINOMI

- Il prodotto di polinomi si esegue con la funzione *conv()*

```
>>p1=[1 3 5]; p2=[1 -2 4];
>> p3=conv(p1,p2)
p3 =
    1    1    3    2   20
```

$p3(x)=x^4 + x^3 + 3x^2 + 2x +20$

- L'operazione duale è la divisione di polinomi, ottenuta con *deconv()*. La sintassi è *[q,r]=deconv(a,b)* che fornisce due polinomi q(x) (quoziente) e r(x) (resto) tali che a(x)=q(x)b(x)+r(x)

POLINOMI

- Le radici di un polinomio si calcolano con *roots()*

```
>>p=[1 9 33 65]
p =
    1    9   33   65

>>roots(p)
ans =
  -5.0000
  -2.0000 + 3.0000i
  -2.0000 - 3.0000i
```

- *polyval(p,x)* calcola il valore p(x) con x dato

```
>> polyval(p,-5)
ans =
    0

>>polyval(p,2.3)
ans =
  200.6770
```


POLINOMI

- Per effettuare la derivata di polinomi sono disponibili le seguenti funzioni
  - *q=polyder(p)*: derivata del polinomio $q(x) = \frac{dp(x)}{dx}$
  - *q=polyder(p1,p2)*: derivata del prodotto $p1(x)\,p2(x)$
  - *[q,d]=polyder(p1,p2)*: derivata del rapporto sotto forma di funzione razionale fratta $\frac{q(x)}{d(x)} = \frac{d}{dx}\frac{p1(x)}{p2(x)}$

POLINOMI

```
>> p=[1 2 3 4];
>>q=polyder(p)
q =
    3    4    3

>> p1=[1 2 3];p2=[4 5 6];
>> q=polyder(p1,p2)
q =

   16   39   56   27
```

```
>> p1=[1 1 1];p2=[1 0 4];
>> [q,d]=polyder(p1,p2)
q =
   -1    6    4

d =
    1    0    8    0   16
```


VISUALIZZAZIONE SCIENTIFICA

- L'integrazione delle funzionalità di calcolo numerico con le elevate capacità grafiche è un punto di forza dell'ambiente MATLAB
- La visualizzazione scientifica è la rappresentazione grafica di dati. Utile nel
  - trovare modelli
  - identificare tendenze
  - comparare informazioni complesse
  - esaminare oggetti che non possono essere esaminati fisicamente

VISUALIZZAZIONE SCIENTIFICA

- Sono forniti un ampio insieme di funzioni ad alto livello per visualizzare dati (2-D, 3-D , movies, etc.)

- Inoltre si ha la possibilità di accedere anche alle proprietà a basso livello degli oggetti grafici (*Handle Graphics*) per gestire completamente la visualizzazione dei dati

GRAFICI

- Per default MATLAB traccia grafici sulla finestra 1; si possono aprire altre finestre di visualizzazione con il comando *figure*.
- Si chiude la finestra corrente con il comando *close*
- Con *figure(n)* si rende corrente la figura numero n e con *close(n)* si chiude la figura numero n

- Per visualizzare grafici bidimensionali (2-D) si utilizza la funzione *plot()*

GRAFICI

- Consideriamo come varia la densità dell'aria ρ (in kg/m^3) al variare della quota h (in km):

h	7	10	15	21	27	34	39	43	47	51
ρ	556	369	191	75	26.2	9.9	4.4	2.3	1.4	0.8

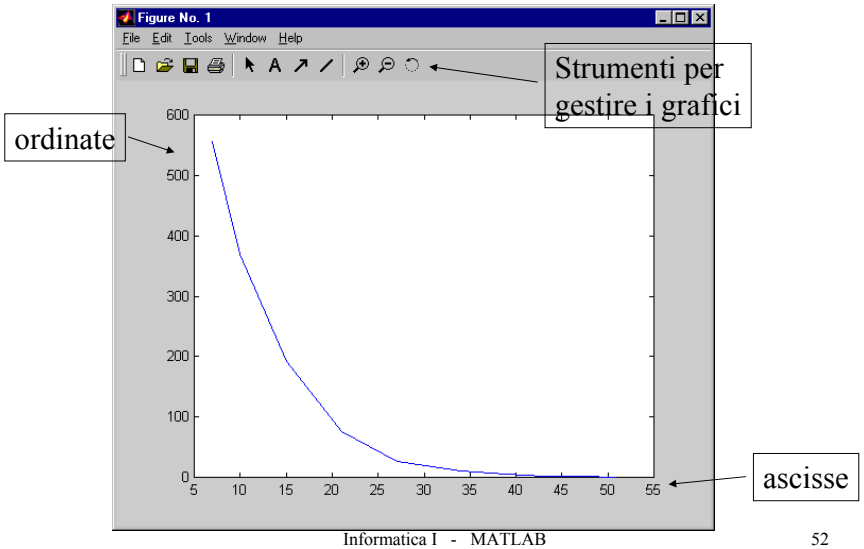
Inseriamo i dati in due vettori

```
>>h = [7   10  15  21  27  34  39  43  47  51];
>>r = [556 369 191 75  26.2 9.9 4.4 2.3 1.4 0.8];
>>plot(h,r)
```

Visualizziamo i dati tramite *plot(ascisse, ordinate)*

GRAFICI

Finestra di visualizzazione



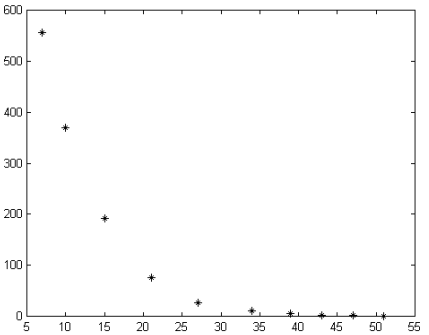
GRAFICI

- La scalatura degli assi è automatica
- Il titolo della finestra (Figure No. 1) indica il numero della finestra stessa
- I punti tracciati sono automaticamente congiunti da linee a tratto pieno
- Non si è dovuto scrivere codice specifico per produrre un output grafico

GRAFICI

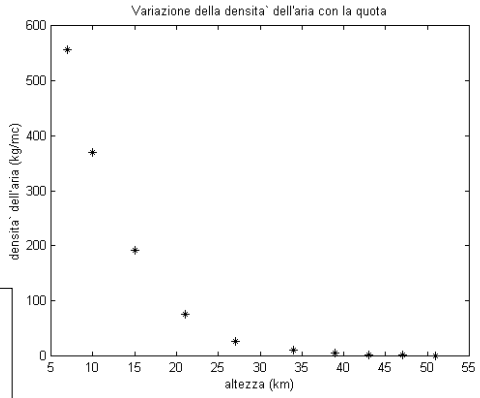
- Congiungere con linee i dati sui grafici è visivamente gradevole, tuttavia è importante indicare anche i dati effettivi: si possono usare parametri opzionali di *plot()*

```
>>plot(h,r,'*')
```



GRAFICI

- Si devono sempre indicare ascisse, ordinate e titolo di un grafico

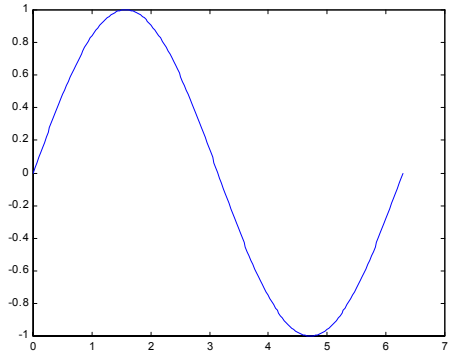



```
>>xlabel('altezza (km)')
>>ylabel('densita` dell''aria
      (kg/mc)')
>>title('Variazione della
    densita` dell''aria con
    la quota')
```


GRAFICI: 2-D

• Per tracciare il grafico di una funzione y=f(t), si deve creare una *tabella di valori della funzione* (come abbiamo visto): specificare un dominio discreto con l'operatore : e calcolare i corrispondenti valori di codominio

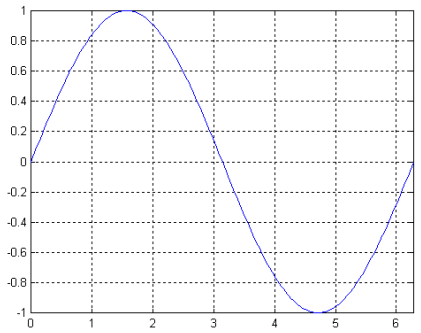
```
>>t=0:pi/100:2*pi;
>> y=sin(t);
>> plot(t,y)
```



GRAFICI : 2-D

- Esistono vari controlli sugli assi
  - axis([xmin xmax ymin ymax])
  - axis square , axis equal
  - axis auto
  - axis on , axis off
  - grid on , grid off

```
>>axis([0  2*pi  -1  1])
>>grid on
```

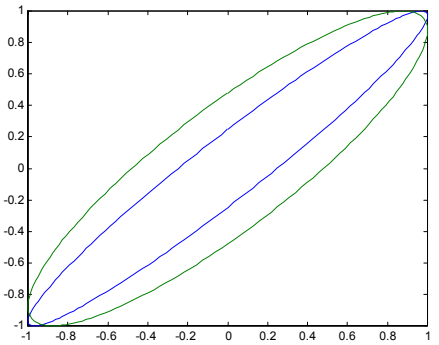


GRAFICI : 2-D

- Si possono visualizzare più grafici insieme

```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1,x,y2)
```

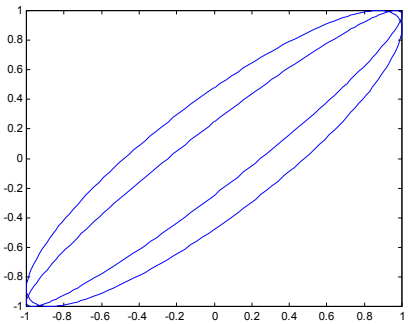
Si devono ripetere ascisse e ordinate



GRAFICI : 2-D

- Si possono aggiungere curve ad un grafico esistente con il comando *hold on*

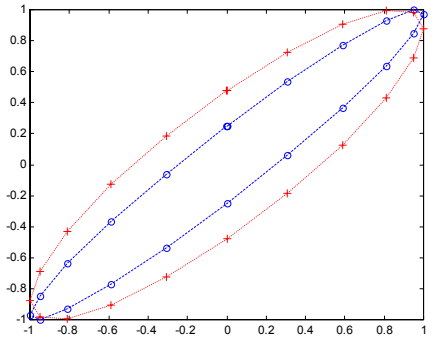
```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1)
>>hold on
>> plot(x,y2)
```



GRAFICI : 2-D

- Si possono differenziare i grafici utilizzando i parametri opzionali di plot()

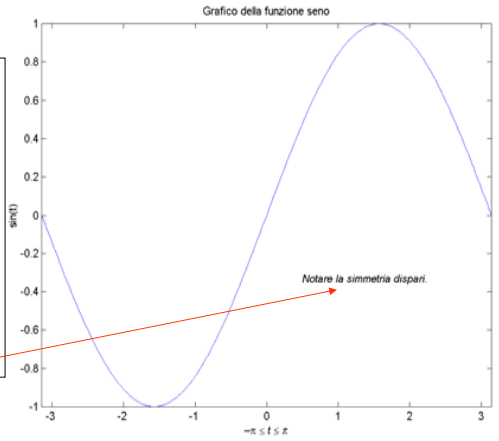
```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1,'b--o')
>>hold on
>> plot(x,y2,'r:+')
```



GRAFICI : 2-D

- Si possono inserire titoli, testi ed etichette (LaTeX)

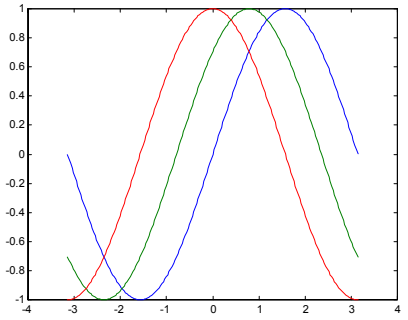
```
>>t = -pi:pi/100:pi;
>> y = sin(t);
>> plot(t,y)
>> axis([-pi pi -1 1])
>> xlabel('-\pi \leq \itt \leq \pi')
>> ylabel('sin(t)')
>> title('Grafico della funzione seno')
>> text(0.5,-1/3,' \it{Notare la simmetria dispari.}')
```



GRAFICI : 2-D

- plot() visualizza le colonne di matrici in ingresso

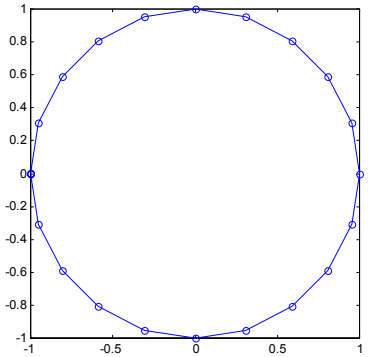
```
>> t = -pi:pi/100:pi;    t=t';
>> a=[sin(t) sin(t+pi/4) sin(t+pi/2)];
>> plot(t,a)
```



GRAFICI : 2-D

- plot() visualizza anche numeri complessi: traccia una curva nel piano complesso (Re{},Im{}). Per esempio la funzione complessa $f(t) = e^{jt}$

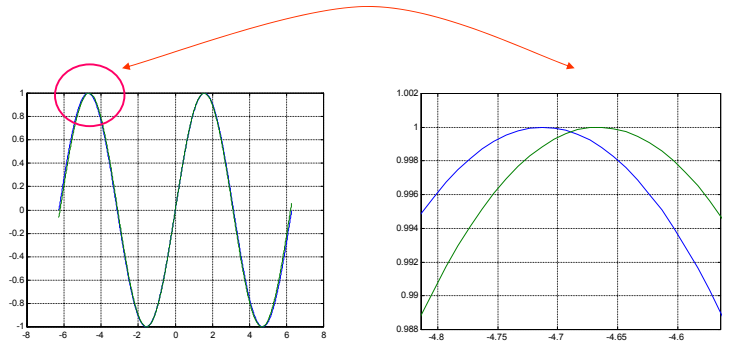
```
>>t = -pi:pi/10:pi;
>>plot(exp(i*t),'-o')
>>axis square
```



GRAFICI : 2-D

- Utile funzione è zoom:

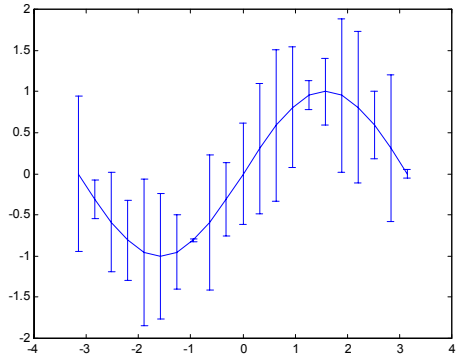
```
>>x=-2*pi:0.01:2*pi;
>>plot(x,sin(x),x,sin(x*1.01))
>> zoom xon, grid on
```



GRAFICI : 2-D

- Vi sono molte altre funzioni specifiche per visualizzare grafici: g.e. errorbar(), bar()

```
>>t = -pi:pi/10:pi;
>>y=sin(t);
>>e=rand(size(t));
>>errorbar(t,y,e)
```

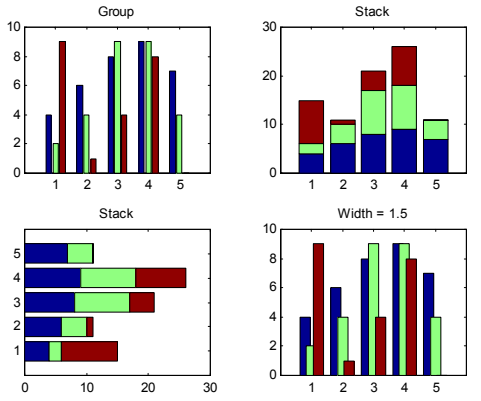


GRAFICI : 2-D

- Per visualizzare più grafici sulla stessa finestra si usa *subplot()*

```
>>figure
>> Y = round(rand(5,3)*10);

>> subplot(2,2,1)
>> bar(Y, 'group')
>> title 'Group'
>> subplot(2,2,2)
>> bar(Y, 'stack')
>> title 'Stack'
>> subplot(2,2,3)
>> barh(Y, 'stack')
>> title 'Stack'
>> subplot(2,2,4)
>> bar(Y,1.5)
>> title 'Width = 1.5'
```



GRAFICI : 2-D

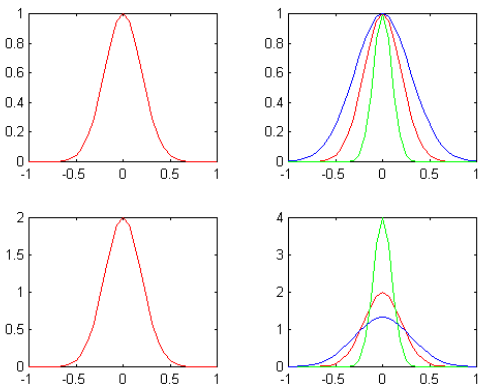
Consideriamo la funzione Gaussiana

$$\begin{cases} g(x) = \exp(-\frac{x^2}{2\sigma^2}) \\ g(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2}) \end{cases}$$

```
x=-1:2/32:1;
figure
subplot(2,2,1)
s=0.2, plot(x,exp(-(x.^2)/(2*s^2)),'r')
subplot(2,2,2)
s=0.2;, plot(x,exp(-(x.^2)/(2*s^2)),'r'),hold on
s=0.1; plot(x,exp(-(x.^2)/(2*s^2)),'g')
s=0.3; plot(x,exp(-(x.^2)/(2*s^2)),'b'),hold off
subplot(2,2,3)
s=0.2; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'r')
subplot(2,2,4)
s=0.2; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'r')
hold on
s=0.1; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'g')
s=0.3; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'b')
hold off
```


GRAFICI : 2-D

$g(x) = \exp(-\frac{x^2}{2\sigma^2})$



red σ=0.2
green σ=0.1
blue σ=0.3

$g(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2})$

GRAFICI: 3-D

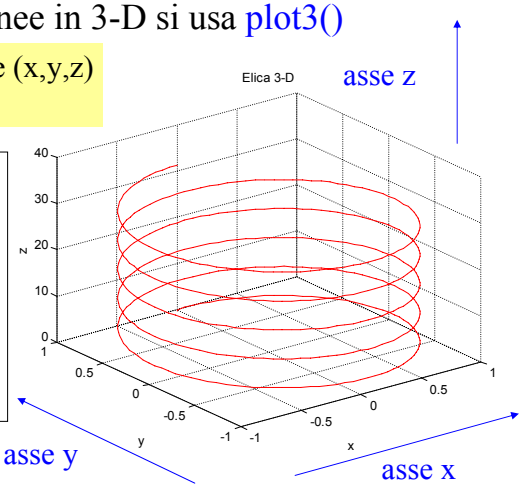
- La *grafica tridimensionale* permette di visualizzare
  - traiettorie (linee) nello spazio tridimensionale
  - superfici generate da funzioni di due variabili f(x,y) o parametriche
  - dati che dipendono da due variabili (e.g. mappe)
  - immagini
  - solidi di rotazione

GRAFICI: 3-D

- Per visualizzare linee in 3-D si usa plot3()

Si specificano le coordinate (x,y,z) dei punti della traiettoria

```
>>t=0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t,'r')
>> grid on
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Elica 3-D')
```



GRAFICI: 3-D

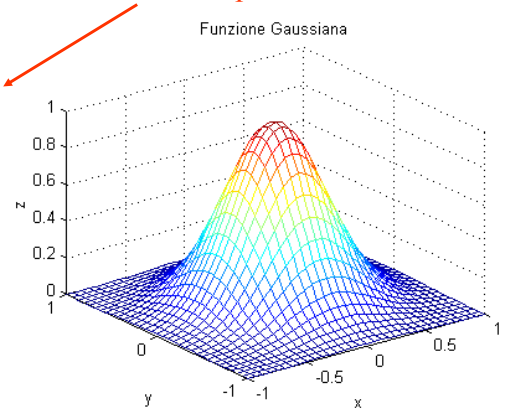
- Anche per funzioni di 2 variabili (*z=f(x,y)*) si deve creare una tabella di valori:
  - si specifica un dominio *discreto* bidimensionale (una griglia su un piano *(x,y)*) con la funzione *meshgrid()*
  - si valuta la funzione *f()* nei valori del dominio *(x,y)*
  - infine si visualizzano i valori calcolati con la funzione *mesh()*

GRAFICI: 3-D

$$z = g(x,y) = \exp(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2})$$

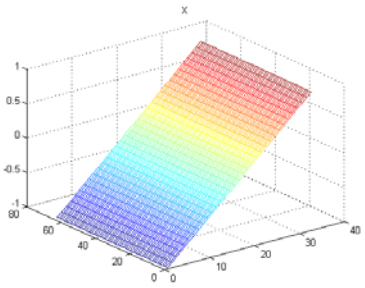
Dominio di campionamento

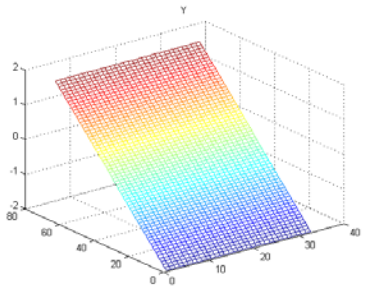
```
>>sx=0.35; sy=0.35;
>>[X,Y]=meshgrid(-1:2/32:1);
>> Z=exp(-(X.^2)/(2*sx^2) -...
(Y.^2)/(2*sy^2));
>> mesh(X,Y,Z)
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Funzione Gaussiana)
```



GRAFICI: 3-D

```
>>[X,Y]=meshgrid(-1:2/32:1,-2:2/32:2);
>>figure, mesh(X),title('X')
>>figure, mesh(Y), title('Y')
```

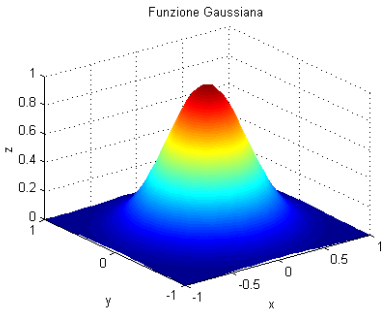




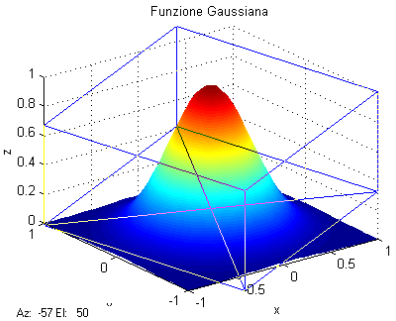
GRAFICI: 3-D

- Esistono molte altre funzioni di visualizzazione (e.g. surf) e di manipolazione (e.g. rotate3d)

```
>>surf(X,Y,Z), shading interp
```

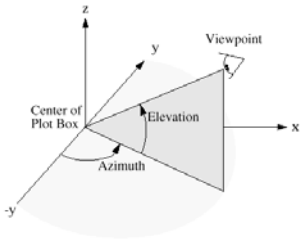



```
>>rotate3d
```

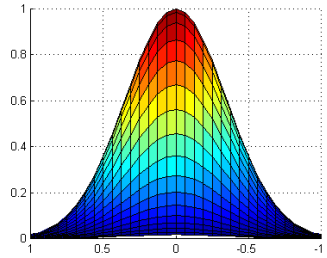


GRAFICI: 3-D

- Si cambia vista di un grafico con view
  - view(azimuth,elevation) o view([x y z])




```
>>surf(X,Y,Z), shading faceted, view([0 1 0])
```

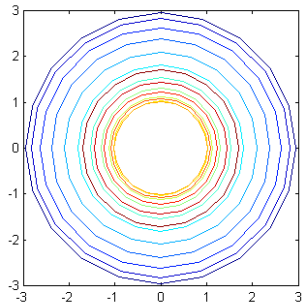


GRAFICI: 3-D

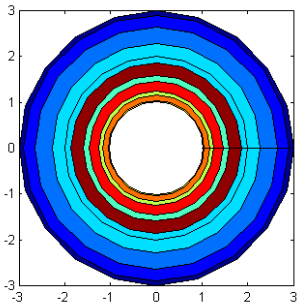
- Consideriamo funzioni che non producono visualizzazioni nello spazio tridimensionale, ma che agiscono comunque su matrici
  - contour(X,Y,Z,n) visualizza curve di livello
  - image(x,y,Z) visualizza dati come immagini
  - pcolor(X,Y,C) visualizza dati come matrici di celle
- Immagini: una griglia di punti il cui valore è codificato con un colore o livello di grigio. Si può considerare una matrice: le righe e colonne formano la struttura spaziale dell'immagine e il valore degli elementi costituiscono il colore

GRAFICI: 3-D

```
>>contour(X,Y,Z,15)
>>axis square
```

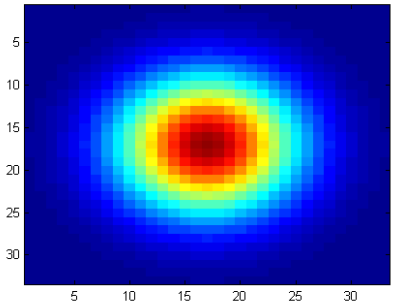


```
>>contourf(X,Y,Z,10)
>>axis square
```

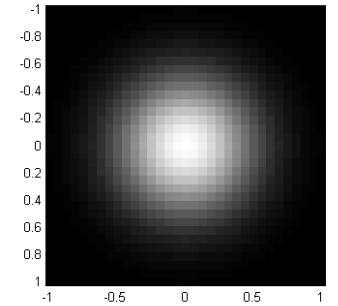


GRAFICI: 3-D

```
>>imagesc(Z)
```



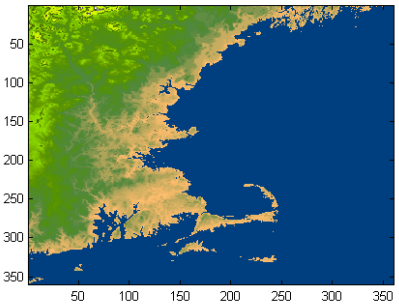
```
>>imagesc(Z)
>>axis square
>>colormap(gray)
```



GRAFICI: 3-D

```
>>load clown
>>imagesc(X)
>>colormap(map)
>>axis off
```

```
>>load cape
>>imagesc(X)
>>colormap(map)
```



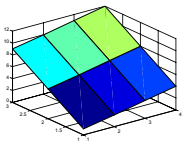
GRAFICI: 3-D

- Attenzione a come le funzioni visualizzano i dati

```
>>a=[1 2 3 4 ;5 6 7 8; 9 10 11 12 ];
>>figure,  surf(a)
>>figure,  imagesc(a)
>>colormap(gray)
>>figure,  pcolor(a)
>>colormap(gray)
>>a= 1   2   3   4
      5   6   7   8
      9  10  11  12
```

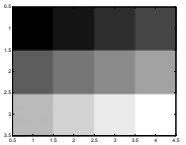
C

R



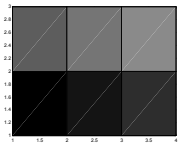
C

R



C

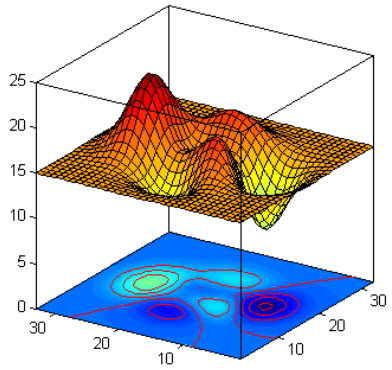
R



GRAFICI: 3-D

- Si possono usare insieme le funzioni viste per ottenere grafici complessi

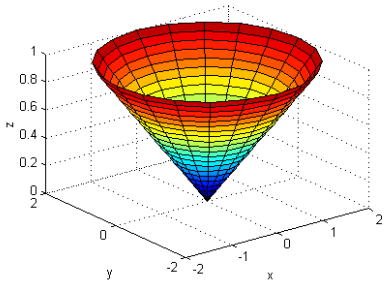
```
>>a=peaks(33);
>> pcolor(a)
>> shading interp
>> axis square
>> hold on
>> contour(a,'r')
>> surf(a+15)
>> view(-57,22)
```



SOLIDI DI ROTAZIONE

- La funzione *cylinder(r)* disegna la superficie di rotazione che ha per generatrice la curva descritta dal vettore *r*: per esempio disegnare il cono generato dalla rotazione della retta $y = x, \; x \in [0,2]$

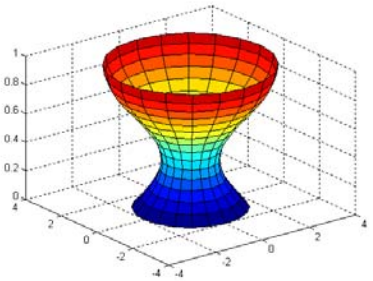
```
>>x=0:0.1:2;
>>y=x;
>>cylinder(y)
>>xlabel('x')
>>ylabel('y')
>>zlabel('z')
```

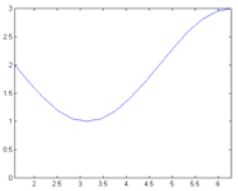


SOLIDI DI ROTAZIONE

- Si può usare una curva per descrivere il profilo

```
>>t = pi/2:pi/10:2*pi;
>>y= 2+cos(t);
>>cylinder(y)
```



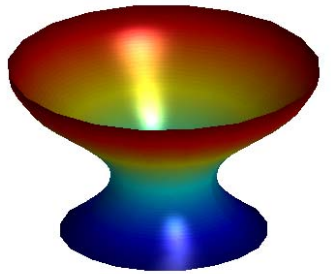


```
>>plot(t,y)
>>axis([pi/2 2*pi 0 3]);
```


SOLIDI DI ROTAZIONE

- Vi sono molte funzioni per manipolare la visualizzazione dei solidi (materiale, luce, angolo di vista)

```
>>axis off
>>shading interp
>>material metal
>>lightangle(45,30)
>>lighting phong
>>view([26 38])
```



HANDLE GRAPHICS

- MATLAB fornisce un insieme di funzioni a basso livello per manipolare elementi grafici: questo sistema è chiamato Handle Graphics
- Gli oggetti grafici sono le primitive di tale sistema
- Ci sono 11 tipi di oggetti Handle Graphics, organizzati in una struttura gerarchica
- Per creare un oggetto è necessario richiamare la funzione corrispondente: *figure, axes, line ...*

GUI

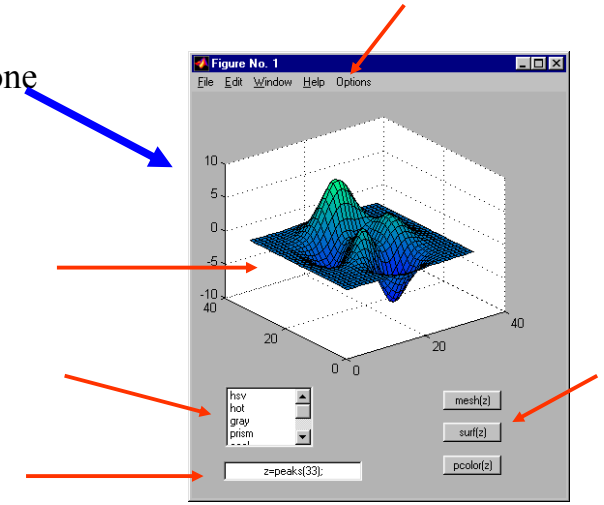
È possibile sviluppare in MATLAB strumenti basati su Graphical User Interface (GUI)

- I principi di un buon progetto di GUI sono generali, anche se si utilizzano strumenti specifici
- In particolare in MATLAB si sviluppa una GUI attraverso Guide (Graphical User Interface Development Environment)

GUI: ESEMPIO

- Esempio di un'applicazione completa.

>>guide



STAMPA DEI GRAFICI

- Con l'opzione Print dal menu File
- Importare in altra applicazione con Copy Figure dal menu Edit
- Salvare in diversi formati con print

24-bit RGB TIFF with packbits compression

>>print -dtiff nome_file

>>print -depsc2 nome_file

Level 2 color Encapsulated PostScript

STRUTTURE DATI

Vediamo alcune strutture dati diverse dalle matrici

- Caratteri e testo
- Cell array
- Strutture
- Array multidimensionali
- Classi

CARATTERI E TESTO

- Per inserire testo si usano gli apici ‘ ‘

```
>>s='Ciao'
s =
  Ciao
>>whos s
  Name      Size         Bytes  Class
  s         1x4              8  char array
Grand total is 4 elements using 8 bytes
```

- È un vettore di caratteri

CARATTERI E TESTO

- Il testo è memorizzato come interi (codice ASCII)

```
>>a=double(s)
a =
    67   105    97   111
>>a(1,1)=99;
>>s=char(a)
s =
  ciao
```

Funzioni di conversione

CARATTERI E TESTO

- Si possono concatenare stringhe

```
>>a=[s '  mondo']
a =
  ciao mondo
>>a=[s ;'  mondo']
??? All rows in the bracketed expression must have the same
number of columns.
```

CARATTERI E TESTO

- Le righe di una matrice devono avere la stessa dimensione: si usa la funzione char()

```
>>s=char('Queste','sono','prove')
s =
Queste
sono
prove
>> size(s)
ans =
    3     6
```


CARATTERI E TESTO

- Si possono convertire stringhe in numeri e viceversa

```
>> a=str2num('123')
a =  123
>> s=num2str(345)
s =  345
>>whos
  Name      Size          Bytes  Class
  a         1x1               8  double array
  s         1x3               6  char array
Grand total is 4 elements using 14 bytes
```


CONTROLLO FLUSSO DATI

- In MATLAB si usano 5 costrutti
  - if
  - switch
  - for
  - while
  - break / return

CONTROLLO FLUSSO DATI: if

```
>>x=1.2;
>>if   x  <  0
>>y = -1;
>>elseif  x  >  0
>>y = 1;
>>else
>>y = 0;
>>end
>>y
y =
    1
```

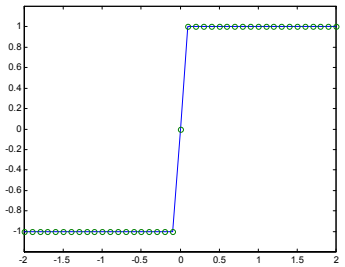
- Ogni istruzione if è terminata da un end
- Non sono necessarie parentesi
- Gli operatori relazionali agiscono sugli array elemento per elemento, quindi per lavorare con array utilizzare le funzioni is* (forniscono lo stato)

CONTROLLO FLUSSO DATI: if

```
>>x=[1.2 1.2];
>>y=[1.2 1.2];
>>x==y
ans =
    1     1

>>isequal(x,y)
ans =
    1
```

```
>>x=-2:0.1:2;   y=zeros(size(x));
>>y =y -1*(x<0);   y =y+ 1*(x>0);
>>y =y+ 0*(x==0);
>>plot(x,y,x,y,'o'),   axis([-2 2 -1.2 1.2])
```



CONTROLLO FLUSSO DATI : switch

```
>>ii=1;
>>switch ( ii )
>>case 0
>>disp('Opzione  0');
>>case 1
>>disp('Opzione  1');
>>otherwise
>>disp('Opzione ???');
>>end
 Opzione  1
```

- Ogni istruzione switch è terminata con end
- Il valore di default è trattato con otherwise
- Quando un case è vero, gli altri sono saltati: non è necessario break.
- La funzione disp permette di visualizzare testi o array

CONTROLLO FLUSSO DATI : for

```
>>for ii=1:3
>>for jj=1:3
>>a(ii,jj)=(ii-1)*3 +jj;
>>end
>>end
>>a
a =
    1    2    3
    4    5    6
    7    8    9
```

- Ogni for è terminato con un end
- Non sono necessarie parentesi
- Si può utilizzare un incremento qualsiasi
  for ii = 1 : -0.3 : -2,   end

N.B In MATLAB è meglio utilizzare codice vettorizzato

CONTROLLO FLUSSO DATI : while

```
>>b=10;  a= -3;
>>while b-a > 0
>>b=b-1;
>>end
>>b
b =
   -3
```

- Ogni while è terminato con un end
- Non sono necessarie parentesi

CONTROLLO FLUSSO DATI : break

```
>>b=10;  a= -3;
>>while b-a > 0
>>if b<=0
>>break
>>end
 >>b=b-1;
>>end
>>b
b =
    0
```

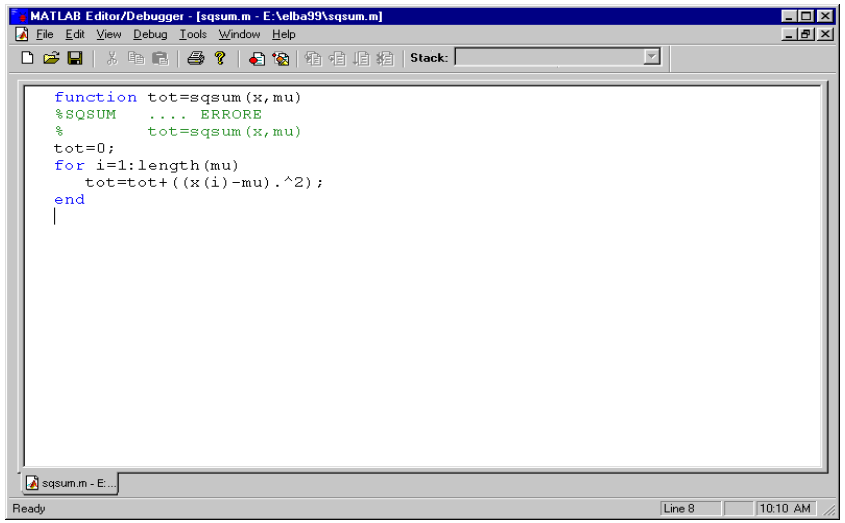
- break permette di uscire dai loop for e while
- Permette di uscire solo dal loop più interno

- return permette di tornare alla funzione chiamante

M-file

- File che contengono codice in linguaggio MATLAB sono chiamati M-file.
- Devono avere estensione .m
- Sono file di testo: possono essere creati con qualsiasi editor ma è preferibile usare quello built-in
- Evidenzia con colori diversi la sintassi e contiene un debugger

M-file



M-file

Script	**Function**
• Non ha argomenti di input ed output	• Ha argomenti di input ed output
• Opera sui dati nello workspace	• Le variabili interne sono locali
• Utile per automatizzare una serie di passi ripetitivi	• Utile per estendere le funzionalità di MATLAB

M-file: SCRIPT

Nome del file → script_2.m

Linea di commento →

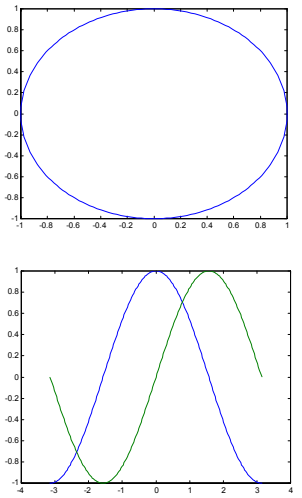
Calcoli →

Controllo flusso dati e visualizzazione →

```
%Esempio di script
x=-pi:pi/100:pi;
ys=sin(x);
yc=cos(x);
if stato
   plot(x,yc,x,ys)
else
   plot(yc,ys)
end
```


M-file: SCRIPT

```
>>script_2
??? Undefined function or variable
    'stato'.
Error in ==> e:\elba99\script_2.m
On line 5  ==> if stato
>>stato=0;
>>script_2
>>stato=1;
>>script_2
>>who
Your variables are:
stato    x        yc       ys
```



M-file: FUNCTION

Nome del file → function_2.m

Definizione di funz. →

Linea di help1 →

Testo di help →

Corpo della funzione →

Calcoli →

Controllo flusso visualizzazione →

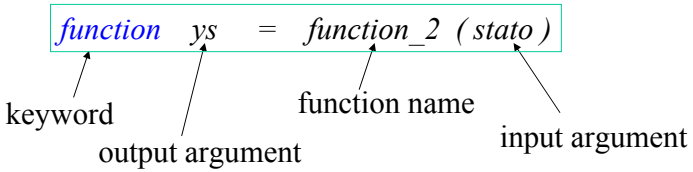
```
function ys = function_2(stato)
%FUNCTION_2  Esempio
%          ys = function_2(stato) : ...
x=-pi:pi/100:pi;
ys=sin(x);
yc=cos(x);
if stato
   plot(x,yc,x,ys)
else
   plot(yc,ys)
end
```


M-file: FUNCTION

```
>>script_2
??? Undefined function or variable
    'stato'.
Error in ==> e:\elba99\script_2.m
On line 5  ==> if stato
>>stato=0;
>>script_2
>>stato=1;
>>script_2
>>who
Your variables are:
stato    x        yc       ys
```

```
>>function_2
??? Input argument 'stato' is undefined.
Error in ==> e:\elba99\function_2.m
On line 7  ==> if stato
>> stato=0;
>>function_2
??? Input argument 'stato' is undefined.
Error in ==> e:\elba99\function_2.m
On line 7  ==> if stato
>>a=function_2(0);
>>a=function_2(1);
>> who
Your variables are:
a        stato
```

M-file: FUNCTION



- Se si hanno molti argomenti

  function [x,y,z] = sphere(theta,phi,rho)

- Se non si hanno argomenti di output

  function sphere(theta,phi,rho)

M-file: FUNCTION

- Quando si chiama una funzione, MATLAB esegue il parsing del codice e produce uno pseudocodice, che alloca in memoria
- Ogni funzione ha un proprio workspace
- Ma possono essere definite variabili globali

M-file: VARIABILI GLOBALI

```
>>global ALPHA
>>ALPHA=3;
>>myp_2([1 2 3])
ans =
    1     8    27
>> ALPHA=2.3;
>>myp_2([1 2 3])
ans =
   1.0000    4.9246   12.5135
```

myp_2.m

```
function y = myp_2(x)
%MYP_2    ....
%         .....
global ALPHA
y=x.^ALPHA;
```


M-file: USER INPUT

- Per ottenere un input da utente durante l'esecuzione di un M-file
  - Visualizzare un prompt e attendere un input (input)
  - Attendere la pressione di un tasto (pause)
  - Costruire una GUI

M-file: USER INPUT

```
>>myp_a_2([1 2 3])
Inserisci l`esponente:3
ans =
    1     8    27
```

myp_a_2.m

```
function y = myp_a_2(x)
%MYP_A_2    ....
%           .....
n=input('Inserisci l`esponente:');
y=x.^n;
```

```
>>myp_a_2([1 2 3])
Inserisci l`esponente:ciao
ans =
ciao
```

```
n=input('Inserisci l`esponente:' , 's');
y=s;
```


M-file: VALUTAZIONE DI STRINGHE

- La funzione eval permette di eseguire una stringa

```
>>eval('a=[1 2 3].^2')
a =
    1     4     9
```

```
>>for ii=1:3
eval([ 'p' , int2str(ii) , '=ii.^2' ])
end

p1 =
    1
p2 =
    4
p3 =
    9
```


M-file: OTTIMIZZAZIONE

- Principalmente ci sono due tecniche
- Vettorizzazione dei loop: cioè sostituire for e while con equivalenti operazioni matriciali
- Preallocazione degli array: allocare la memoria prima di utilizzare gli array

M-file: OTTIMIZZAZIONE

```
>>mybench_2
elapsed_time =
  204.1000
elapsed_time =
    0.0500
```

mybench_2.m

```
%esempio
N=30000;   tic
i=0;  c=zeros(1,N);
for t=0:pi/N:2*pi
   i=i+1;
   c(i)=sin(t);
end
toc,  tic
t=0:pi/N:2*pi;
c=sin(t);  toc
```

M-file: OTTIMIZZAZIONE

```
>>mybench_1_2
elapsed_time =
   5.4300

elapsed_time =
  14.7800
```

Mybench_1_2.m

```
%esempio
N=500;  tic,   a=zeros(N);
for ii=1:N
  for jj=1:N
    a(ii,jj)=ii+jj;
  end
end
toc,  tic
for ii=1:N
  for jj=1:N
    b(ii,jj)=ii+jj;
  end
end,  toc
```


DEBUGGING

Il debugging è il processo che permette di trovare errori nel proprio codice

- Errori di sintassi: li indica MATLAB al prompt
- Errori runtime: errori algoritmici, quindi rilevabili solo da risultati inattesi, perchè interni allo workspace delle funzioni

DEBUGGING

Per isolare gli errori runtime

- Togliere selettivamente i ;
- Rendere la funzione uno script
- Usare l'istruzione keyboard
- Usare il MATLAB debugger

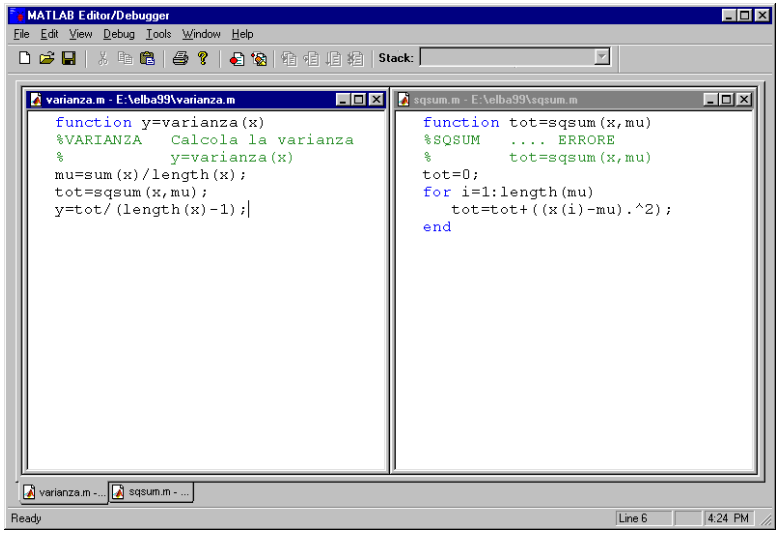
DEBUGGING

```
>>who
>>niente
K» who
Your variables are:
x        y
K» y
y =  4
K» x=3;
K» return
y =  9
>>
```

niente.m

```
function niente()
%       ....
%       ....
x=2;
y=x.^2;
keyboard
y=x.^2
```

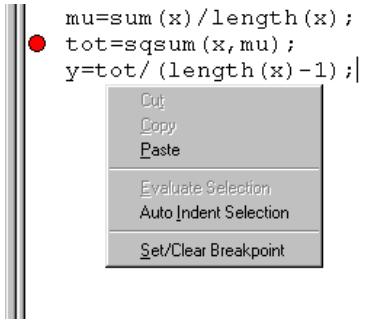
DEBUGGING



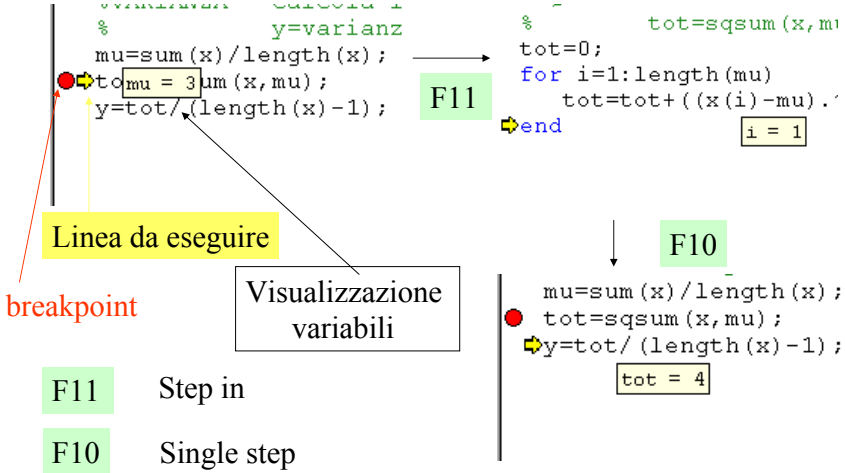
DEBUGGING

```
>> v=[1 2 3 4 5];
>>var1=std(v).^2
var1 =
   2.5000

>>var2=varianza(v)
var2 =
    1
```



DEBUGGING



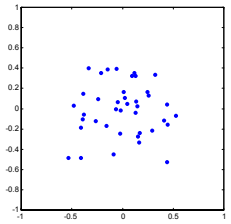
APPLICAZIONI

- Vediamo come MATLAB lavora con particolari tipi di dati e funzioni
  - animazioni
  - I/O dati
  - funzioni di funzione

APPLICAZIONI: ANIMAZIONI

anim_2.m

```
%Brownian motion
n=40; s=.02;
x=rand(n,1)-0.5;  y=rand(n,1)-0.5;
h=plot(x,y,'.');
axis([-1 1 -1 1]),  axis square,  grid off
set(h,'EraseMode','xor','MarkerSize',18)
while 1
  x=x + s*randn(n,1);
  y=y + s*randn(n,1);
  set(h,'XData',x,'YData',y)
  drawnow
end
```



- Adatta per lunghe sequenze di semplici plot
- Per fermare la simulazione digitare <ctrl>-c

I/O

- Oltre alle funzioni save e load MATLAB fornisce funzioni di I/O C-like:
  - fopen, fclose : apertura e chiusura file
  - fscanf, fprintf: lettura e scrittura dati formattati
  - fread, fwrite: lettura e scrittura binaria di dati

I/O FORMATTATO

```
%...
x = 0:.1:1;y = [x; exp(x)];
fid =fopen('exp.txt','w');
fprintf(fid,'%6.2f%12.8f\n',y);
fclose(fid)
```

File su disco

exp.txt

```
0.00    1.00000000
0.10    1.10517092
…
1.00    2.71828183
```

```
%...
fid = fopen('exp.txt');
a = fscanf(fid,'%g %g',[2 inf]);
   % Ora sono due righe
a = a';
fclose(fid)
```

```
>>a
a = 0.00    1.00000000
    0.10    1.10517092
    ...
    1.00    2.71828183
```


I/O BINARIO

```
%...
a=[1 2 3 4; 5 6 7 8];
fid =fopen('a.bin','wb');
fwrite(fid,a,'float');
fclose(fid)
```

a.bin

File su disco

```
%...
fid =fopen('a.bin','rb');
a=fread(fid,[2,4],'float');
fclose(fid)
```

```
>>a
a =
    1    2    3    4
    5    6    7    8
```

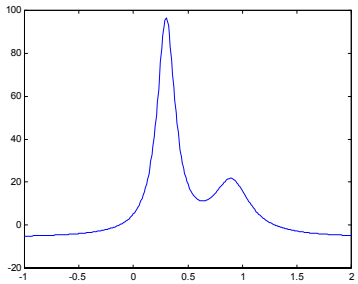
FUNZIONI DI FUNZIONE

Consideriamo un esempio che può essere generalizzato come modello di interfaccia a particolari funzioni di MATLAB

- Integrazione numerica
- Minimo di una funzione
- Equazioni differenziali ordinarie

FUNZIONI DI FUNZIONE

```
>>x=-1:.01:2;
>>plot(x,es_2(x))
```



es_2.m

```
function y=es_2(x)
%ES_2    Esempio
%        y=es_2(x)
y=1./((x -.3).^2 + .01)+...
   1./((x - .9).^2 + .04) -6;
```


INTEGRAZIONE NUMERICA

```
>>quad('es_2',0,1)
ans =
   29.8583

>>quad8('es_2',-10,10)
ans =
  -73.2779
```

Stringa contenente il nome di una funzione

API

- Sebbene MATLAB è un ambiente completo per programmare, esiste la possibilità di interfacciare MATLAB con programmi esterni attraverso la Application Program Interface (API)
- Si può
  - utilizzare MATLAB da programmi C, MATLAB engine
  - utilizzare proprie applicazioni C come funzioni buil-in, MEX-file