Laboratorio di Fisica Computazionale

Carlo Pierleoni

19 aprile 2009

Capitolo 1

Errori

Durante il corso impareremo ad usare il computer uno strumento divenuto ormai indispensabile nel lavoro quotidiano del fisico. Il suo utilizzo appropriato richiede la comprensione di alcune semplici nozioni sul suo funzionamento, alcune delle quali sono state già affrontare nel corso di Laboratorio di Calcolatori I. Per i nostri scopi è importante rivedere brevemente il modo in cui il computer tratta i numeri e le operazioni tra di essi. Poichè la soluzione numerica di un problema sarà sempre necessariamente approssimata, è importante sapere la natura dell'errore che si sta commettendo. Le sorgenti di possibili errori sono:

1. **Errori di troncamento**
   con questo termini si intendono gli errori inerenti al metodo numerico scelto per risolvere un dato problema. Il termine deriva dall'errore che si commette troncando un sviluppo in serie di Taylor allorquando si vuole calcolare il valore di una funzione. Per esempio possiamo approssimare le funzione e^x con la cubica

   $$p_3(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \tag{1.1}$$

   ma sappiamo che il valore vero di e^x si ottiene solo sommando infinite potenze

   $$e^x = p_3(x) + \sum_{n=4}^{\infty} \frac{x^n}{n!} \tag{1.2}$$

   Quindi la stima di e^x tramite la cubica $p_3(x)$ fornisce sono un'approssimazione. E' importante stimare l'errore che si commette. Tuttavia questo errore dipende unicamente dall'algoritmo (ossia dalla schema scelto per il calcolo) e non da come il computer combina i numeri per ottenere il risultato numerico voluto. Un errore di questo tipo può essere ridotto a piacere sommando un numero sempre maggiore di termini. Tuttavia è impossibile sommare infiniti termini e quindi la soluzione sarà sempre approssimata. E' quindi necessario, nella soluzione numerica di problemi schematizzati da equazioni, stabilire l'entità dell'errore che si è disposti ad accettare anche in base al budget (in termini di tempo di CPU) a disposizione.

2. **Errore di arrotondamento**
   Qualsiasi strumento di calcolo rappresenta i numeri con una precisione fissata. I computers non sono da meno e usano generalmente numeri in virgola mobile e una parola di lunghezza fissata. Questo permette di rappresentare tutti i numeri in un range finito di valori e con una precisione finita. Due numeri che differiscono per meno della precisione saranno a tutti gli effetti identici per il nostro computer. Questo tipo di errore è detto di arrotondamento. L'utilizzatore potrà generalmente scegliere se l'ultima cifra decimale deve essere arrotondanta o troncata (qual'è la differenza?).

3. **Errore di programmazione**
   I moderni computers possono commettere errori nella manipolazione delle istruzioni ed è successo per esempio con la prima versione del processore Pentium II. Tuttavia questo tipo di errori sono molto rari, altrimenti non potremmo considerare il computer uno strumento affidabile. Molto più frequente è invece l'errore di programmazione (detto *bug* nel gergo informatico). Questo può essere di tipo logico, ossia l'algoritmo scelto non è in realtà appropriato alla soluzione del problema, oppure sintattico, ossia la traduzione dell'algoritmo in linguaggio di programmazione non è corretta. La gran parte del tempo umano che occorre per produrre un codice viene spesa per trovare questo tipo di errori (processo di *debug*, ossia eliminazione dei bugs). La soluzione di questo tipo di errori risiede nell'attenzione del programmatore e nella sua accuratezza nella scrittura dei codici (vi sono certe regole fondamentali che, pur non necessarie, rendono un codice molto più sicuro e facile da debuggare). L'abilità del programmatore risiede, non solo nella capicità di scrittura di un codice, ma soprattutto nella definizione dei tests necessari per verificare il suo corretto funzionamento.

4. **Errori propagati**
   Questo tipo di errore è più sottile dei precedenti. Per errore propagato si intende l'errore nei passi successivi di un processo che è causato da un errore precedentemente commesso. Questo si aggiunge all'errore che si commette al passo in questione. Questo tipo di errore è a volte difficile da valutare a priori e può essere di importanza critica. Infatti se l'effetto di un errore viene amplificato nel corso della procedura, la soluzione che si ottiene sarà sempre più imprecisa all'avanzare del processo e potrà non avere più nulla in comune con la vera soluzione del problema.

1.1 Aritmetica in virgola mobile e stima degli errori

Nella lista degli errori su riportata, l'unico intrinseco alla natura del computer è quello di arrotondamento. Per capire i dettagli di tale errore è necessario capire come le quantità numeriche sono rappresentate in un computer. I numeri sono generalmnte rappresentati nella notazione a **virgola mobile** (floating-point). Non tutti i computers usano la stessa tecnica ma la procedura generale è simile. Possiamo rappresentare un numero floating-point nella forma

$$\pm f B^e = \pm .d_1 d_2 \dots d_p B^e \tag{1.3}$$

dove

- d_i sono cifre (o *digits*) con $0 \leq d_i \leq B-1$
- B è la base usata (generalmente 2,16 o 10)
- p è il numero di cifre significative, ossia la precisione
- e è l'esponente intero, definito in un dato intervallo $e \in [-E_{min}, E_{max}]$
- $f = d_1 d_2 \dots d_p$ è la parte frazionaria del numero

Nelle calcolatrici tascabili la base è generalmente 10, nei computer di solito è 2 ma altre basi, ad esempio 16, sono anche usate. Vediamo alcuni esempi: supponiamo B=10 e p=4. I numeri sulla sinistra saranno rappresentati come nella colonna di sinistra:

$$\begin{aligned} 27.39 &\rightarrow +.2739 \times 10^2 \\ -0.00124 &\rightarrow -.124 \times 10^{-2} \\ 37000 &\rightarrow +.37 \times 10^5 \end{aligned}$$

dove abbiamo richiesto che la prima cifra $d_1 \neq 0$. Tali numeri si dicono normalizzati. Come già detto la quantità di numeri distinti in un dato intervallo è necessariamente finita. Si può mostrare che il numero di numeri floating-point normalizzati che si possono rappresentare con la codifica (**??**) è

$$2(B-1)B^{p-1}(B^{E_{max}} + B^{Emin} + 1) + 1 \tag{1.4}$$

Due livelli di standard, decisi dall'IEEE (Institute of Electrical and Electronics Engineers), per rappresentare i numeri floating-point sono usati nei computers

- formato **Single-precision** (32 bits): 1 bit per il segno, 8 bits per l'esponente, 23 bits per le cifre significative. B=2, $e \in [-128, 127]$, p=23
- formato **Double-precision** (64 bits): 1 bit per il segno, 11 bits per l'esponente, 52 bits per le cifre significative. B=2, $e \in [-1024, 1023]$, p=52

Il primo bit rappresenta il segno del numero. Il segno dell'esponente è rappresentato shiftando il valore dell'esponente di 128 (o 1024) in modo da usare solo numeri senza segno da 0 a 255 (o 2047). Il numero di valori distinti secondo la (**??**) è quindi

- single precision: $2 * 1 * 2^{22} * (2^7 + 2^7) + 1 \simeq 2.14 \times 10^9$
- double precision: $2 * 1 * 2^{51} * (2^{10} + 2^{10}) + 1 \simeq 9.11 \times 10^{18}$

È interessante esaminare la distribuzione dei numeri floating point normalizzati. Consideriamo il caso B=2, p=2 con $-2 \leq e \leq 3$. Tutti i numeri normalizzati saranno della forma

$$\pm .10 * 2^e \qquad o \qquad \pm .11 * 2^e \qquad\qquad -2 \leq e \leq 3 \tag{1.5}$$

Poichè per le frazioni binarie si ha: $.10 = 1/2, .11 = 1/2 + 1/4 = 3/4$, questi numeri vanno da -6 a +6 ($.11 * 2^3 = 3/4 * 2^3 = 6$). Quindi la lista dei numeri positivi è:

$$\begin{array}{lll} .10 * 2^{-2} = \dfrac{1}{8} & .10 * 2^{-1} = \dfrac{1}{4} & .10 * 2^0 = \dfrac{1}{2} \\ .10 * 2^1 = 1 & .10 * 2^2 = 2 & .10 * 2^3 = 4 \\ .11 * 2^{-2} = \dfrac{3}{16} & .11 * 2^{-1} = \dfrac{3}{8} & .11 * 2^0 = \dfrac{3}{4} \\ .11 * 2^1 = \dfrac{3}{2} & .11 * 2^2 = 3 & .11 * 2^3 = 6 \end{array}$$

In questo piccolo sistema floating point ci sono solo 25 numeri diversi (compreso lo zero) e quindi mappiamo l'intervallo [-6,6] su 25 numeri distinti.

Per semplificare l'analisi dell'accuratezza nella operazioni floating-point, è più facile considerare un sistema di numerazione in base 10 usando numeri floating-point normalizzati. L'analisi nelle altre basi numeriche è analoga. Per semplificare l'analisi, assumiamo solo 3 cifre significative e una sola cifra per l'esponente. Abbiamo quindi: $B = 10, p = 3, e \in [-9, 9]$. Vediamo la differenza tra arrotondamento e troncamento. Quando sottraiamo o addizioniamo due numeri floating-point, le cifre decimali del numero con esponente più piccolo devono essere spostate per allineare le posizioni decimali tra i due numeri. Infine, a seconda del risultato, la somma può essere spostata e l'esponente aggiustato di conseguenza per ottenere la normalizzazione. Per esempio:

$$.137 \times 10^1 + .269 \times 10^{-1}$$
$$.137 \times 10^1$$

$+ \quad .00269 \times 10^1 \qquad (allineamento)$

$\_\_\_\_\_$

$.13969 \times 10^1 \qquad \rightarrow troncato = .139 \times 10^1$

$+ \quad .0005$

$\_\_\_\_\_$

$.14019 \times 10^1 \qquad \rightarrow arrotondato = .140 \times 10^1$

Altro esempio:

$.485 \times 10^4 - .482 \times 10^4$

$.485 \times 10^4$

$- \quad .482 \times 10^4 \qquad (allineamento)$

$\_\_\_\_\_$

$.003 \times 10^4$

$.300 \times 10^2 \qquad normalizzato \rightarrow troncato = .300 \times 10^2$

$+ \quad .0005$

$\_\_\_\_\_$

$.3005 \times 10^2 \qquad \rightarrow arrotondato = .300 \times 10^2$

Osserviamo che l'arrotondamento richiede una operazione in più e quindi è più costoso che il troncamento. Notiamo inoltre la perdita di precisione nel secondo esempio che avviene sempre quando sottraiamo due numeri dello stesso ordine di grandezza. Anche se in apparenza il risultato ha ancora 3 cifre decimali, i due zeri non sono significativi perchè sono stati aggiunti in fase di normalizzazione e non sono il risultato dell'operazione eseguita.Questo problema è l'origine principale di perdita di accuratezza nelle operazioni tra numeri floating-point.
La moltiplicazione di due numeri a n cifre significative fornisce un numero con 2n cifre significative. Il registro floating-point di un computer normalmente permette l'operazione e converte poi il risultato in un numero con n cifre significative. Lo stesso vale per le divisioni.Vediamo alcuni esempi:

$.403 \times 10^6 * .197 \times 10^{-1}$

$.403 \qquad 6 \qquad$ (moltiplica decimali e somma gli esponenti)

$* \quad .197 \qquad -1$

$\_\_\_\_\_ \qquad \_\_\_\_\_$

$.079391 \qquad 5$

$.79391 \times 10^4 \qquad normalizzato \rightarrow troncato = .793 \times 10^4$

$+ \quad .0005$

$\_\_\_\_\_$

$.79441 \times 10^4 \qquad \rightarrow arrotondato = .794 \times 10^4$

$.356 \times 10^{-1} / .156 \times 10^4$

$.356 \qquad -1 \qquad$ (divide decimali e sottrae gli esponenti)

$\div \quad .156 \qquad 4$

$\_\_\_\_\_ \qquad \_\_\_\_\_$

$$
\begin{array}{lll}
 & 2.28205 \times 10^{-5} & \\
 & 0.228205 \times 10^{-4} & normalizzato \rightarrow troncato = .228 \times 10^{-4} \\
+ & .0005 & \\
 & \text{----------} & \\
 & .228705 \times 10^{-4} & \rightarrow arrotondato = .228 \times 10^{-4}
\end{array}
$$

Le operazione di moltiplicazione e divisione sono molto più lente di somma e sottrazione. Il tempo impiegato per la moltiplicazione è da 2.5 a 10 volte maggiore di quello per la somma o sottrazione. La divisione è ancora più lenta con un tempo da 4 a 25 volte maggiore. Questo quando tali operazioni sono eseguite via hardware (coprocessore matematico). Altrimenti l'esecuzione via software è ancora più lenta. Questo era vero per i primi computers fino agli anni '80. Nei moderni computers, la presenza di potenti coprocessori matematici ha ridotto considerevolmente questi fattori e moltiplicazioni e divisioni richiedono circa lo stesso tempo di CPU di somme e sottrazioni.

Poichè la lunghezza della parola utilizzata dal computer è finita, l'operazione di convertire un numero nella base interna del computer introduce un certo errore.Infatti dobbiamo mappare un numero infinito di numeri reali in un insieme discreto di numeri manipolati dal computer. Nell'esempio fatto con tre cifre decimali abbiamo solo 900 diversi valori per le cifre significative per ogni potenza della base. Questo significa che l'insieme continuo dei numeri in ciascuna decade deve essere mappato su questi 900 numeri frazionari. Questo fatto introduce quindi una risoluzione finita nella rappresentazione di un numero nel computer. Il numero zero è un caso speciale tra i floating-points. Non può essere normalizzato perché tutti i suoi bits sono generalmente nulli. Inoltre l'esponente è generalmente il più negativo possibile per evitare di dover shiftare gli altri numeri nelle somme. Nel nostro esempio in base 10 a zero corrisponde il numero $\pm 0.1 * 10^{-9}$. Se si cerca di rappresentare un numero minore di questo si incorrerà in un *exponent underflow*. Il comportamento del computer davanti ad una situazione del genere dipende in genere dal compilatore e può essere fissato tramite delle istruzioni i fase di compilazione del programma. In genere l'azione di default è quella di settare il numero a zero e continuare l'esecuzione. In modo analogo ogni tentativo di rappresentare numeri più grandi di $0.999 * 10^9$ genera un *exponent overflow*. A meno di aver specificato altrimenti, davanti a questo tipo di evento il computer termina l'esecuzione del programma con un errore.
Bisogna sempre tenere a mente che a volte le regole che abbiamo imparato dell'aritmetica continua non si applicano all'aritmetica dei numeri floating-point. Tipici esempi: sommando 1000 volte il numero 0.001 si può non ottenere 1.000 esattamente; multiplicare un numero per 1 non sempre fornisce lo stesso numero di partenza; in una somma di molti addendi il risultato dipende dall'ordine in cui si eseguono le operazioni.

1.2 Errori assoluto e relativo, cifre significative

Definiamo l'*errore assoluto* come la differenza tra il valore vero e quello approssimato e quindi si ottiene il valore vero sommando a quello approssimato il suo errore. Se l'errore assoluto viene diviso per il valore vero otteniamo l'*errore relativo*. Quando il valore esatto è zero l'errore relativo è indefinito.
Oltre ai concetti di errore assoluto e relativo abbiamo anche usato quello di cifre significative. Supponiamo che sia:

$$\text{Valore vero :} \qquad d_1, d_2, \ldots, d_n, d_{n+1}, \ldots, d_p \tag{1.6}$$

$$\text{valore approssimato:} \qquad d_1, d_2, \ldots, d_n, e_{n+1}, \ldots, e_p \tag{1.7}$$

con $d_1 \neq 0$ e la prima differenza tra i due numeri è all'$n+1$-esima cifra decimale. Diremo che il valore vero e quello approssimato sono in accordo fino all'n-esima cifra significativa se $|d_{n+1} - e_{n+1}| < 0.5$. Altrimenti diremo che l'accordo è fino all'$n-1$-esima cifra decimale.

1.3 Esempio di laboratorio: calcolo della funzione esponenziale

Si tratta di scrivere un programma (in un linguaggio di programmazione scelto) che, per un dato valore della variabile x, calcoli il valore numerico di e^x con la precisione scelta. Per far questo questo si utilizza lo sviluppo in serie di Taylor della funzione

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{1.8}$$

che in pratica deve essere necessariamente troncato

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} \sim p_N(x) = \sum_{n=0}^{N} \frac{x^n}{n!} \tag{1.9}$$

La questione è in che modo decidere per quale valore N fermarci. Possiamo decidere di volere la massima precisione compatibile con la scelta della lunghezza di parola che stiamo usando per il calcolo. In questo caso dobbiamo confrontare il valore ottenuto con $n+1$ termini con il valore per n termini ed arrestare il calcolo non appena $|p_{n+1}(x) - p_n(x)| = 0$ dove per 0 si intende l'exception underflow (ossia il numero più piccolo che la macchina è in grado di trattare). Alternativamente possiamo decidere di fissare la precisione del nostro calcolo ad un valore finito ϵ, e fermare la procedura non appena $|p_{n+1}(x) - p_n(x)| < \epsilon$ (errore assoluto), o meglio $|[p_{n+1}(x) - p_n(x)]/p_n(x)| < \epsilon$ (errore relativo).

Un aspetto da curare nella realizzazione di un programma è quello di cercare di minimizzare il numero di operazioni necessarie per ottenere il risultato voluto e in particolare di cercare uno schema logico che renda minimo il numero di moltiplicazioni e divisioni, come abbiamo visto le operazioni più costose tra quelle semplici. Per esempio per un polinomio di ordine 4

$$p_4(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \tag{1.10}$$

posso usare la seguente scrittura

$$p_4(x) = a_0 + x\{a_1 + x[a_2 + x(a_3 + a_4 x)]\} \tag{1.11}$$

L'uso della prima forma richiede l'esecuzione di 14 moltiplicazioni e 4 addizioni mentre nella seconda forma sono sufficienti 4 moltiplicazioni e 4 somme e la sua esecuzione può risultare notevolmente più rapida, specialmente se ripetura per molti valori diversi della variabile x (come succede ad un qualunque programma di grafica quando deve disegnare la curva corrispondente al polinomio assegnato per valori della variabile all'interno di un certo intervallo assegnato).

Torniamo al caso specifico della funzione esponenziale e supponiamo di avere optato per la massima precisione possibile. Notiamo che se chiamiamo a_n l'n-esimo termine della somma, abbiamo la relazione ricorsiva: $a_{n+1} = a_n x/(n+1)$. Scriveremo allora uno schema del genere

1. definizione ed inizializzazione delle variabili
2. lettura del valore dell'ascissa x
3. ciclo ricorsivo
   - i=i+1

- a=a*x/i
- sum=sum+a
- controllo sulla precisione per terminare il ciclo

4. scrittura del risultato e termine dell'esecuzione

Uno schema come questo funziona bene finchè il valore dell'ascissa è positivo. In questo caso il risultato si ottiene come somma di addendi tutti positivi e non abbiamo perdita di precisione. Se invece $x < 0$ e magari in valore assoluto grande, la somma è a segni alterni e, come abbiamo visto nella sezione precedente nell'operazione di somma in generale abbiamo una perdita di precisione. Possiamo verificare questo fatto se proviamo ad usare un tale programma per calcolare $e^{-30.4}$. Sul G4 in precisione semplice si ottiene $e^{-30.4} = 91887.2188$ dopo 86 iterazioni, un risultato palesemente insensato. Usando invece la doppia precisione si ottiene -0.000227170954 dopo 119 iterazioni, risultato anch'esso insensato (perché?).
La soluzione a questo problema è semplice: basta ricordare che

$$e^{-|x|} = (e^{|x|})^{-1} = \frac{1}{e^{|x|}}$$

e quindi ogni qualvolta $x < 0$ calcolare $e^{|x|}$ e poi prendere il suo reciproco.
Questo stratagemma ci assicura di ottenere sempre un risultato accurato ma a che costo? Possiamo trovare un algoritmo che converga più rapidamente? La risposta è affermativa e basta osservare che

$$e^x = e^{[x]}e^y$$

dove abbiamo indicato con $[x]$ la parte intera di x e con $y = x - [x]$ la sua parte frazionaria. Essendo $[x]$ un numero intero (positivo o negativo) il primo esponenziale si riduce all'elevamento a potenza del numero di Nepero e il cui valore numerico, ottenuto precedentemente con uno dei programmi su scritti fissando $x = 1$, sarà stato salvato in memoria. Rimane quindi da calcolare e^y con $-1 < y < 1$ il che ci assicura una rapida convergenza della serie di potenze.

Capitolo 2

Equazioni non lineari

In questo capitolo affronteremo dei metodi per trovare le soluzioni di equazioni algebriche non lineari o equazioni trascendenti. Questo in partica significa individuare il valore dell'incognita per cui l'equazione assegnata, pensata in termini di funzione di x, incontra l'asse delle ascisse. Tutti i metodi di soluzione di questo problema sono iterativi, ossia partendo da un valore iniziale approssimato della soluzione cercata, forniscono un nuovo valore più vicino alla soluzione vera. Questo nuovo valore è poi usato come valore iniziale per ottenere un'altra stima della soluzione re-iterando la procedura. Se le stime che si ottengono alle varie iterazioni approssimano la soluzione vera con sempre maggiore accuratezza, si dice che lo schema converge, se l'accuratezza (ossia l'errore relativo) non aumenta nel corso dell'iterazione lo schema è stabile ma non converge, se infine l'accuratezza diminuisce (ossia l'errore aumenta) lo schema è instabile. Per ogni dato schema è chiaramente importante capire a priori le sue proprietà di convergenza. Altra considerazione di carattere generale, a volte legata alla stabilità dell'algoritmo, è la scelta dei punti (o del punto) iniziali. È ovvio che una condizione necessaria affinchè la funzione abbia uno zero all'interno di un certo intervallo è che assuma segni diversi agli estremi dell' intervallo. Ciò tuttavia non garantisce di avere una sola soluzione, ma di averne in numero dispari. Sarà quindi necessaria una preliminare ricerca grossolana per isolare una soluzione. Questa operazione viene chiamata *bracketing* della soluzione. Un modo semplice, ma non automatico, per eseguire il bracketing è semplicemente di tracciare il grafico della funzione con un programma di grafica e scegliere un opportuno intervallo che contenga un solo zero. In seguito vedremo una procedura automatica che permette di eseguire il bracketing senza l'intervento umano.

2.1 Metodo di bisezione

Consideriamo l'equazione algebrica di terzo grado

$$f(x) = x^3 + x^2 - 3x - 3 = 0$$

dalla figura **??**, ottenuta con xmgrace, possiamo vedere che una radice è compresa tra x=1 a x=2. In particolare f(1)=-4 e f(2)=3. Scegliamo quindi come intervallo iniziale $[1, 2]$. Come prima cosa consideriamo il punto di mezzo $x = 1.5$. Poichè $f(1.5) < 0$ la soluzione sarà nell'intervallo $[1.5, 2]$. Questo schema può essere iterato e l'intervallo si restringerà di un fattore 2 ad ogni iterazione finchè non diventa inferiore alla precisione voluta (che può essere quella della macchina). Una importante caratteristica del metodo di bisezione è la conoscenza immediata dell'errore nella stima della soluzione. Infatti il valore esatto dello zero sarà contenuto nell'ultimo intervallo considerato. Questo metodo può essere usato anche per equazioni trascendenti tipo $f(x) = e^x - 3x = 0$.

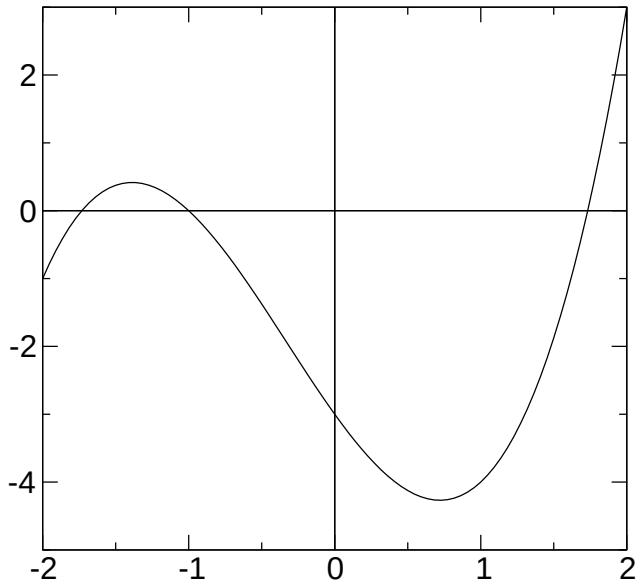

Figura 2.1: grafico della cubica $f(x) = x^3 + x^2 - 3x - 3$

2.2 Metodo dell'interpolazione lineare

Il metodo della bisezione è facile da implementare, robusto (nel senso che fornisce la soluzione quasi sempre), e l'analisi dell'errore è semplice. Tuttavia non è molto efficiente. Per aumentare il tasso di convergenza possiamo usare il *metodo dell'interpolazione lineare*. L'idea è di approssimare la funzione $f(x)$ nell'intervallo $[x_1, x_2]$ con la retta

$$r(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) \tag{2.1}$$

che si annulla per

$$x_3 = x_1 - f(x_1)\frac{x_2 - x_1}{f(x_2) - f(x_1)} \tag{2.2}$$

Si calcola quindi il valore $f(x_3)$ e si sceglie l'intervallo in cui f(x) cambia segno, $[x_1, x_3]$ o $[x_3, x_2]$. Questo sarà il nuovo intervallo per la prossima iterazione.
Il metodo dell'interpolazione lineare converge in genere più rapidamente del semplice metodo di bisezione. Naturalmente sarà tanto più rapido quanto più la funzione in esame è ben rappresentata da una retta, ossia quanto più la sua derivata prima è costante nell'intervallo considerato. Nel caso considerato della cubica con intervallo iniziale $[1, 2]$ il metodo di bisezione impiega 23 iterazioni per convergere alla soluzione mentre con l'interpolazione lineare sono necessarie solo 9 iterazioni.
Un inconveniente del metodo dell'interpolazione lineare è la sua convergenza unilaterale alla soluzione (provare per credere). Se la funzione ha una notevole curvatura nell'intervallo scelto la convergenza può risultare lenta. Per rimediare a questo inconveniente si può sostituire il valore della funzione nel punto che non cambia con la sua metà ad ogni iterazione. Questa divisione va però fatta solo se non abbiamo cambiato intervallo.

2.3 Metodo di Newton

Come il precedente, anche il metodo di Newton è basato su un'approssimazione lineare della funzione, ma utilizza la tangente alla curva ossia la derivata della funzione. Partendo da una posizione iniziale non troppo lontana dalla soluzione, si estrapola lungo la tangente alla curva in quella posizione fino ad incontrare l'asse delle ascisse. Si prende il valore dell'intercetta come condizione iniziale per la prossima iterazione. Si avrà:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad\qquad n = 1, 2, 3, \ldots \tag{2.3}$$

In generale questo metodo converge più velocemente dei precedenti. Mostreremo in seguito che il metodo converge quadraticamente, ossia che l'errore che si commette all'iterazione i è il quadrato dell'errore al passo precedente per una costante moltiplicativa: $\epsilon_i = K\epsilon_{i-1}^2$. Il risultato è che il numero di cifre significative di accuratezza si raddoppiano ad ogni iterazione. L'inconveniente è tuttavia il dover calcolare la funzione e la sua derivata ad ogni iterazione.
Occorre sottolineare che in alcuni casi il metodo di Newton non converge come vedremo nel seguito quando mostreremo che quando converge lo fa quadraticamente.

2.4 Metodo di Muller

Nel metodo di Muller invece di un'approssimazione lineare alla funzione si usa una approssimazione quadratica, ossia si approssima la funzione localmente con una parabola. Si costruisce una parabola fittando la funzione in 3 punti vicini alla soluzione cercata (come nel metodo della bisezione bisogna individuare un intervallo che comprenda lo zero e poi si prende un terzo punto). Supponiamo di aver scelto x_1, x_2 tali che $x_1 < x_2$ e $f(x_1) * f(x_2) < 0$ e prendiamo un terzo punto $x_1 < x_0 < x_2$ (per esempio il punto di mezzo). La generica parabola $ay^2 + by + c$ con $y = x - x_0$ che passa per questi tre punti ha come coefficienti la soluzione del sistema

$$\begin{aligned} c &= f(x_0) = f_0 \\ ay_1^2 + by_1 + c &= f(x_1) = f_1 \\ ay_2^2 + by_2 + c &= f(x_2) = f_2 \end{aligned} \tag{2.4}$$

La prima equazione fornisce direttamente $c = f_0$ e risolvendo il rimanente sistema di 2 equazioni lineari accoppiate, per esempio con Cramer, si ottiene

$$a = \frac{f_0}{y_1 y_2} + \frac{f_1 y_2 - f_2 y_1}{y_1 y_2 (y_1 - y_2)} \tag{2.5}$$

$$b = \frac{y_1^2 f_2 - y_2^2 f_1}{y_1 y_2 (y_1 - y_2)} - \frac{f_0 (y_1 + y_2)}{y_1 y_2} \tag{2.6}$$

Una volta trovati i coefficienti della parabola si trovano le soluzioni $y_\pm = (-b \pm \sqrt{b^2 - 4ac})/2a$ di $ay^2 + by + c = 0$ e si prende quella più vicina a x_0, ossia quella che ha modulo più piccolo (y_+ se $b > 0$, y_- se $b < 0$). La nuova soluzione diventa il punto x_0 per la seconda iterazione e gli estremi dell'intervallo che la contengono i nuovi punti x_1, x_2.
Il metodo di Muller ha un tasso di convergenza simile a quello di Newton ma non richiede il calcolo della derivata. Inoltre ad ogni iterazione, dopo la prima, è richiesto un solo computo della funzione. Questo metodo può essere molto utile nel caso in cui il calcolo della funzione e della sua derivata è molto costoso.

2.5 Metodo dell'iterazione

Riscriviamo l'equazione originale $f(x) = 0$ nella forma $x = g(x)$. Per certe condizioni particolari (da derivare) possiamo trovare la soluzione che soddisfa $x = g(x)$ con il metodo iterativo

$$x_{n+1} = g(x_n) \qquad\qquad n = 1, 2, 3, \ldots \tag{2.7}$$

che, se lo schema converge, tende alla soluzione voluta per $n \to \infty$. Consideriamo ad esempio l'equazione di secondo grado $f(x) = x^2 - 2x - 3 = 0$ che ha soluzioni $x = 3$ e $x = -1$. L'equazione di partenza si può riscrivere come $x = \sqrt{2x+3}$ (per $x > -3/2$) e quindi avremo $g_1(x) = \sqrt{2x+3}$. Oppure si può ottenere $x = g_2(x)$ con $g_2(x) = 3/(x-2)$. Infine si può anche riscrivere come $x = g_3(x)$ con $g_3(x) = (x^2-3)/2$. Fissando il punto iniziale a $x = 4$ e iterando lo schema con le tre diverse scritture si può vedere che: 1) $x_{n+1} = g_1(x_n)$ converge in maniera monotona alla soluzione $x = 3$; 2) $x_{n+1} = g_2(x_n)$ converge in maniera oscillante ma alla soluzione $x = -1$; 3) $x_{n+1} = g_3(x_n)$ diverge.
Se uno schema di questo tipo converge, il punto asintotico che soddisfa la $x = g(x)$ si chiama punto fisso perchè una volta raggiunto questo punto lo schema lì resta. Studiamo ora le condizioni per la convergenza dello schema $x_{n+1} = g(x_n)$. Chiamiamo x_f il punto fisso dello schema. Possiamo riscrivere

$$x_{n+1} - x_f = g(x_n) - g(x_f) = \frac{g(x_n) - g(x_f)}{(x_n - x_f)}(x_n - x_f) \tag{2.8}$$

Se $g(x)$ e $g'(x)$ sono continue in $[x_f, x_n]$ possiamo invocare il teorema del valore medio e scrivere

$$x_{n+1} - x_f = g'(\xi_n)(x_n - x_f) \tag{2.9}$$

con $\xi_n \in [x_n, x_f]$. Se definiamo l'errore commesso all'iesime iterazione con $\epsilon_i = x_i - x_f$ allora possiamo riscivere

$$\epsilon_{n+1} = g'(\xi_n)\epsilon_i \tag{2.10}$$

Considerando i moduli possiamo riscrivere $|\epsilon_{n+1}| = |g'(\xi_n)||\epsilon_i|$. Supponiamo ora che $|g'(x)| \leq K < 1$ per x in un intorno di x_f di ampiezza h. Se x_1 è scelto in tale intervallo, anche x_2 vi cadrà e l'algoritmo sarà convergente poichè

$$|\epsilon_{n+1}| \leq K|\epsilon_i| \leq k^2|\epsilon_{i-1}| \leq \ldots \leq K^n|\epsilon_1| \tag{2.11}$$

Quindi riassumendo possiamo dire che *se $g(x)$ e $g'(x)$ sono continue in un intorno di un punto fisso x_f dell'equazione $x = g(x)$, e se $|g'(x)| < 1$ per tutti gli x in quell'intorno, allora $x_{n+1} = g(x_n)$ converge a x_f se x_1 è stato scelto in quell'intorno. Notare che questa è solo condizione necessaria.*
È ovvio che il tasso di convergenza è regolato dal modulo della derivata nell'intorno considerato ed è grande per funzioni con modulo della derivata prima molto inferiore a 1, mentre è piccolo se il modulo della derivata prima è vicina a 1. Man mano che le iterazioni si avvicinano a x_f, il valore di $g'(\xi_n)$ tende a $g'(x_f)$, ossia a una costante. In questo limite l'errore al passo n-esimo diventa proporzionale a quello al passo precedente. Per questo il metodo è detto anche di *iterazione lineare*.

2.6 Convergenza del metodo di Newton

Siamo ora in grado di derivare un criterio di convergenza per il metodo di Newton. Lo schema di Newton

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad\qquad n = 1, 2, 3, \ldots \tag{2.12}$$

è della forma $x_{n+1} = g(x_n)$. Quindi è convergente se

$$|g'(x)| = \left|1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}\right| = \left|\frac{f(x)f''(x)}{[f'(x)]^2}\right| < 1 \tag{2.13}$$

in un intorno del punto fisso x_f. La condizione è solo necessaria e richiede come al solito la continuità di f(x) e delle sue derivate prima e seconda. Inoltre $f'(x) \neq 0$ in questo intorno.
Dimostriamo ora che il metodo di Newton converge quadraticamente. Per $x = x_f$ si ha $x_f = g(x_f)$ e possiamo quindi scrivere

$$x_{n+1} - x_f = g(x_n) - g(x_f) \tag{2.14}$$

Espandiamo ora in serie di potenze di $(x_n - x_f)$ la $g(x_n)$

$$g(x_n) = g(x_f) + g'(x_f)(x_n - x_f) + \frac{g''(\xi)}{2}(x_n - x_f)^2 \tag{2.15}$$

con $|\xi| < |x_n - x_f|$. Notando che $g'(x_f) = [f(x_f)f''(x_f)]/[f'(x_f)]^2 = 0$ perchè $f(x_f) = 0$, si ottiene

$$\epsilon_{n+1} = x_{n+1} - x_f = g(x_n) - g(x_f) = \frac{g''(\xi)}{2}\epsilon_n^2 \tag{2.16}$$

che mostra come l'errore all'iterazione $(n+1)$-esima sia proporzionale al quadrato dell'errore all'iterazione n-esima[1]. Naturalmente la relazione quadratica vale sufficientemente vicino alla convergenza, ossia quando si può considerare la derivata seconda costante ed uguale al valore assunto in x_f.

[1]Se $f'(x_f) = 0$ si può dimostrare che la convergenza è solo lineare

Capitolo 3

Sistemi di equazioni

In questo capitolo desciveremo alcuni metodi semplici per risolvere sistemi di equazioni (lineari e non) accoppiate. Useremo la notazione matriciale che risulta essere la descizione naturale di questi problemi. Vediamo quindi un breve ripasso sulle matrici e loro proprietà.

3.1 Matrici: principali definizioni e richiami

Una matrice è un arrangiamento rettangolare di numeri, in generale complessi. La taglia della matrice è data dal numero di righe e di colonne in cui sono sistemati i numeri, detti gli elementi della matrice. La posizione degli elementi nella matrice è individuata da due indici, il primo indica la riga e il secondo la colonna.
Due matrici di stessa taglia possono essere sommate e sottratte. La matrice risultante avrà come elementi la somma o la differenza degli elementi con stessi indici delle due matrtici originali

$$\mathbf{C} = \mathbf{A} \pm \mathbf{B} \qquad c_{ij} = a_{ij} \pm b_{ij} \qquad \forall (i,j)(i = 1, n; j = 1, k) \tag{3.1}$$

Una matrice $A_{n\times m}$ può essere moltiplicata righe per colonne con una matrice $B_{m\times k}$ e il risultato sarà una matrice $C_{n\times k}$ con elementi definiti da

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \qquad c_{ij} = \sum_{l=1}^{m} a_{il} b_{lj} \qquad \forall (i,j)(i = 1, n; j = 1, k) \tag{3.2}$$

In generale il prodotto tra matrici non gode della proprietà commutativa ossia $AB \neq BA$. Un eccezione a questa regola si ha per matrici diagonali.
Se una matrice è moltiplicata per uno scalare la matrice risultante ha come elementi il prodotto dello scalare per l'elemento corrispondente della matrice originale:

$$\mathbf{C} = k\mathbf{A} \qquad c_{ij} = k a_{ij} \qquad \forall (i,j)(i = 1, n; j = 1, k) \tag{3.3}$$

Una matrice con una sola colonna $(n \times 1)$ è un vettore colonna a n componenti, mentre una matrice con una sola riga $(1 \times m)$ è un vettore riga a m componenti. Ovviamenti gli elementi di un vettore sono individuati da un solo indice. La moltiplicazione di una matrice per un vettore segue le stesse regole della moltiplicazione di due matrici. Quindi una matrice $A_{n\times m}$ può essere moltiplicata a destra per un vettore colonna di dimensione m generando un vettore colonna a n componenti, oppure può essere moltiplicata a sinistra per un vettore riga di dimensione n generando un vettore riga a m componenti

$$\mathbf{b}_{n\times 1} = \mathbf{M}_{n\times m} \cdot \mathbf{a}_{m\times 1} \qquad b_i = \sum_{k=1}^{m} m_{ik} a_k \qquad \forall i = 1, n \tag{3.4}$$

$$\mathbf{b}_{1\times m} = \mathbf{a}_{1\times n} \cdot \mathbf{M}_{n\times m} \qquad b_j = \sum_{k=1}^{n} a_k m_{kj} \qquad \forall j = 1, m \tag{3.5}$$

La definizione della moltiplicazione righe per colonne ci permette di scrivere un sistema di n equazioni lineari in m incognite in termini matriciali

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = b_2 \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nm}x_m = b_m \end{aligned} \tag{3.6}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3.7}$$

Ad ogni matrice $\mathbf{A}_{n\times m}$ è associata la sua *trasposta* $\mathbf{A}^T_{m\times n}$ ottenuta da $\mathbf{A}$ scambiando le righe con le colonne. In altri termini vale la relazione

$$(\mathbf{A}^T)_{ij} = (\mathbf{A})_{ji} = a_{ji} \tag{3.8}$$

Ad una matrice con elementi complessi, oltre alla trasposta si associa l'*hermitiana coniugata*, ottenuta dalla matrice originale con una operazione di trasposizione e di coniugazione complessa (le due operazioni commutano tra loro)

$$\mathbf{A}^\dagger \equiv (\mathbf{A}^T)^* \qquad (\mathbf{A}^\dagger)_{ij} = a^*_{ji} \qquad \forall(i,j) \tag{3.9}$$

Dato un vettore colonna a n componenti $\mathbf{a}$, il suo *trasposto* $\mathbf{a}^T$ è un vettore riga. Il prodotto scalare di due vettori $\mathbf{a}$ e $\mathbf{b}$ a n elementi è uno scalare ottenuto dalla moltiplicazione matriciale tra il vettore riga $\mathbf{a}^T$ e il vettore colonna $\mathbf{b}$

$$s = \mathbf{a}^T \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i \tag{3.10}$$

Si definisce *norma euclidea* (o modulo quadrato) del vettore il prodotto

$$\mathbf{a}^\dagger \cdot \mathbf{a} = \sum_{i=1}^{n} (\mathbf{a}^\dagger)_i (\mathbf{a})_i = \sum_{i=1}^{n} a^*_i a_i = \sum_{i=1}^{n} |a_i|^2 \tag{3.11}$$

Consideriamo ora matrici quadrate ossia matrici con ugual numero di righe e di colonne (ad un sistema di n equazioni in n incognite è associata una matrice quadrata $n \times n$). Alcuni tipi speciali di matrici quadarate sono:

- diagonale: in cui solo gli elementi sulla diagonale principale sono non nulli ($a_{ij} \neq 0 \Longleftrightarrow i = j$).
- triagolare superiore: in cui tutti gli elementi sotto la diagonale principale sono nulli ($a_{ij} \neq 0 \Longleftrightarrow i \geq j$)
- triangolare inferiore: in cui tutti gli elementi sopra la diagonale principale sono nulli ($a_{ij} \neq 0 \Longleftrightarrow i \leq j$)
- tridiagonale: in cui solo gli elementi della diagonale principale e quelli nelle posizioni adiacenti alla diagonale principale sono non nulli ($a_{ij} \neq 0 \Longleftrightarrow j = i, i \pm 1$)

Per una matrice quadrata possiamo definire due quantità scalari, la *Traccia* e il *Determinante*. La traccia di una matrice quadrata è definita dalla somma dei sui elementi diagonali

$$Tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} \tag{3.12}$$

mentre il determinante è costruito a partire da una qualunque riga o colonna considerando la somma algebrica degli elementi della riga per il loro minore corrispondente. Il minore associato ad un dato elemento di matrice a_{ij} è il determinante della matrice che si ottiene eliminando dalla matrice originale la riga i-esima e la colonna j-esima. Il segno del minore è positivo se $i+j$ è un numero pari, mentre è negativo se $i+j$ è dispari. È facile convincersi che il determinante di una matrice diagonale è il prodotto degli elementi sulla diagonale. Il calcolo del determinante di una matrice di notevole dimensione è un'operazione che può risultare molto laboriosa. Se riusciamo a scrivere la matrice in forma triangolare allora è il semplice risultato di una moltiplicazione.

3.2 Eliminazione di Gauss

È il padre di tutti i metodi cosiddetti *diretti* per la soluzione di sistemi di equazioni lineari. È la semplice estensione del metodo usato per eliminare una delle incognite in una coppia di equazioni. Consideriamo il seguente esempio

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 2 \end{aligned} \tag{3.13}$$

In termini matriciali possiamo scrivere la seguente matrice aumentata

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 1 & 2 & 3 & 11 \\ 2 & -2 & -1 & 2 \end{array}\right] \tag{3.14}$$

Moltiplicando la prima riga per 1/3 e sottraendola alla seconda si elimina l'elemento $_{21}$. Allo stesso modo, moltiplicando la prima riga per 2/3 e sottraendola alla terza si elimina l'elemento $_{31}$.

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & 7/3 & 7/3 & 7 \\ 0 & -4/3 & -7/3 & -6 \end{array}\right] \tag{3.15}$$

Il passo successivo consiste nell'eliminazione dell'elemento 32 moltiplicando la seconda riga per $-4/7$ e sottraendola alla terza

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & 7/3 & 7/3 & 7 \\ 0 & 0 & -1 & -2 \end{array}\right] \tag{3.16}$$

La terza equazione fornisce direttamente $x_3 = 2$ e risolvendo a ritroso la seconda fornisce $x_2 = 1$ e la prima $x_1 = 3$.

Possiamo costruire una matrice triangolare inferiore $\mathbf{L}$ con i coefficienti usati per la riduzione nelle posizioni che sono state da loro azzerate e con tutti uno sulla diagonale principale. Nel nostro caso abbiamo

$$\mathbf{L} = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & -4/7 & 1 \end{array}\right] \tag{3.17}$$

È facile verificare che, chiamata $\mathbf{A}$ la matrice originale e $\mathbf{U}$ la matrice che si ottiene dopo l'eliminazione di Gauss vale la relazione

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} \tag{3.18}$$

Fattorizzare una matrice come prodotto di una matrice triangolare inferiore e una matrice triangolare superiore è detto *decomposizione LU*. Data una matrice A la decomposizione LU non è unica (vedi dopo).

La seconda fase del metodo consiste nel sostituire a ritroso la soluzione per l'ultima incognita nella penultima equazione, ottenere la penultima incognita e sostituire ultima e penultima incognita nella terz'ultima equazione e così via. Questa perta si chiama *back-substitution*. In pratica per la back-substitution si può procedere combinando le righe in modo da eliminare gli elementi sopra la diagonale e ridursi ad una forma diagonale. Moltiplicando la terza riga per -7/3 e sottraendola alla seconda, e moltiplicando la terza riga per -2 e sottraendola alla prima si ottiene

$$\left[\begin{array}{ccc|c} 3 & -1 & 0 & 8 \\ 0 & 7/3 & 0 & 7/3 \\ 0 & 0 & -1 & -2 \end{array}\right] \tag{3.19}$$

Moltiplicando la seconda riga per -3/7 e sottraendola alla prima si arriva alla forma diagonale

$$\left[\begin{array}{ccc|c} 3 & 0 & 0 & 9 \\ 0 & 7/3 & 0 & 7/3 \\ 0 & 0 & -1 & -2 \end{array}\right] \tag{3.20}$$

Infine dividendo ciascuna riga per l'elemento diagonale corrispondente si ottiene la soluzione voluta del sistema di equazioni. Infatti la matrice si riduce all'identità e il termine noto alla soluzione cercata. Quest'ultima operazione è detta **riduzione**.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{array}\right] \tag{3.21}$$

Bisogna notare che non sempre questa semplice procedura può essere utilizzata. Se ad un certo stadio della triangolarizzazione incontriamo uno zero sulla diagonale, non possiamo usare questa riga per proseguire nella procedura. Si può tuttavia cercare di invertire le righe e continuare il processo (**pivoting**). Il vero problema si incontra se dopo aver ottenuto una matrice triangolare superiore uno degli elementi sulla diagonale è zero. Ricordando che per una matrice triangolare il determinante è il prodotto degli elementi sulla diagonale e che del resto, per una generica matrice quadrata, il determinante è sempre dato dal prodotto degli autovalori, ne ricaviamo che gli elementi diagonali di una matrice triangolare sono anche gli autovalori della matrice. Se una matrice ha un autovalore nullo, il suo determinante è nullo e la matrice non ammette matrice inversa, e quindi non può essere diagonalizzata né tantomeno essere ridotta all'identità. In questo caso il sistema non ha soluzione unica perchè almeno una delle equazioni dipende linearmente dalle altre e il numero delle incognite è maggiore di quello delle equazioni indipendenti.

3.2.1 Considerazioni numeriche

Prima di applicare l'algoritmo appena descritto nel calcolo numerico dobbiamo fare alcune considerazioni basate sulla precisione finita del computer. Per problemi di piccole dimensioni lo schema su illustrato è perfettamente legittimo. Tuttavia la vera utilità del computer risiede nella trattazione di sistemi di molte equazioni la cui soluzione a mano richiederebbe un numero di operazioni enorme. In questo caso le operazioni di moltiplicazione nel corso dell'eliminazione di Gauss possono dare luogo a numeri molto grandi che potrebbero esaurire il registro di memoria del nostro computer e generare un *exception overflow*. Per evitare ciò ordineremo all'inizio le righe in modo da avere come prima riga quella che ha il primo elemento più largo tra tutte le righe. Nel corso dell'eliminazione sottrarremo all'i-esima riga la prima moltiplicata per a_{i1}/a_{11}.

Altro problema in cui si può incorrere durante la soluzione numerica è la divisione per zero. Infatti lo zero può essere generato sulla diagonale nel corso dell'eliminazione anche se la matrice originale non ne conteneva. Una strategia utile per evitare (se possibile) di dividere per zero è quella di arrangiare ad ogni passo dell'eliminazione le righe in modo da avere il coefficiente più grande (in modulo) sulla diagonale. Questa procedura è detta **pivoting**. Il pivoting è parziale se ci limitiamo ad arrangiare le righe, completo se arrangiamo sia le righe che le colonne. Quest'ultimo è raramente usato perchè richiede di tenere in memoria gli scambi delle colonne per ottenere la soluzione voluta. Il pivoting parziale garantisce di evitare la divisione per zero se il sistema ammette soluzione. Se al contrario il sistema non ammette soluzione allora bisogna far uso di algoritmi diversi e più complessi (Singular Value Decomposition, SVD vedi Numerical Recipes). L'operazione di pivoting non solo evita la divisione per zero ma in genere aumenta l'accuratezza della soluzione trovata (vedi dopo). Gli elementi diagonali che si ottengo con il pivoting sono detti *pivots (perni)*.

Illustriamo queste idee e i problemi numerici che si incontrano risolvendo con 4 cifre decimali il semplice esempio già affrontato precedentemente. Consideriamo la matrice aumentata

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 1 & 2 & 3 & 11 \\ 1 & -2 & -1 & 2 \end{array}\right] \tag{3.22}$$

Sottraendo alla riga 2 la riga 1 moltiplicata per 1/3=0.3333, e alla riga 3 la riga 1 moltiplicata per 2/3=0.6667 azzeriamo la prima colonna ottenendo

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ (0.3333) & 2.333 & 2.334 & 7.004 \\ (0.6667) & -1.334 & -2.332 & -5.992 \end{array}\right] \tag{3.23}$$

Sottraendo alla riga 3 la riga 2 moltiplicata per $-1.334/2.333 = -0.5711$ si riduce la matrice in triangolare superiore

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ (0.3333) & 2.333 & 2.334 & 7.004 \\ (0.6667) & (-0.5711) & -1.000 & -1.993 \end{array}\right] \tag{3.24}$$

Tra parentesi abbiamo messo gli elementi della matrice triangolare inferiore che fattorizza la matrice originale. Da questo sistema con la *back substitution* otteniamo $x_3 = 1.993; x_2 = 1.008; x_1 = 3.007$ e la differenza con la soluzione esatta è dovuta all'errore di round-off. Questo effetto è sempre presente e cresce al crescere del numero di operazioni necessarie per l'eliminazione ossia con la dimensione della matrice. Per matrici molto grandi, l'errore di round-off può rovinare completamente una soluzione numerica.

Riassumendo l'algoritmo per l'eliminazione Gaussiana è formato dalle seguenti operazioni

1. Aumentare la matrice $n \times n$ dei coefficienti con il vettore dei termini noti in modo da ottenere una matrice $n \times (n+1)$

2. Scambiare, se necessario, le righe in modo da avere

$$a_{11} = \max_{i \in [1,n]} a_{i1}$$

3. Annullare il primo elemento di tutte le righe da 2 a n sottraendo alla i-esima riga la prima riga moltiplicata per a_{i1}/a_{11}. Memorizzare a_{i1}/a_{11} nella posizione di memoria occupata da a_{i1}.

4. Ripetere i passi 2 e 3 per annullare gli elementi sotto la diagonale principale delle colonne dalla seconda alla $(n-1)$-esima. Per annullare la k-esima colonna, prima scambiare le righe da k a n in modo da avere
$$a_{kk} = \max_{l\in[k,n]} a_{lk}.$$
Poi usare la k-esima riga ed annullare gli elementi della colonna k per eliminazione ossia sottraendo alla riga l-esima la riga k moltiplicata per a_{lk}/a_{kk}. Mettere a_{lk}/a_{kk} nella locazione di memoria occupata predentemente da a_{lk}. Quando $k = n-1$ si otterrà un matrice triangolare superiore. Il prodotto degli elementi sulla diagonale sarà il determinante della matrice originale con il segno positivo se il numero di scambi di riga applicati è pari, con il segno negativo se il numero di inversioni usate è dispari.

5. Ottenere x_n dall'n-esima equazione
$$x_n = \frac{a_{n,n+1}}{a_n}$$

6. Ottenere $x_{n-1}, x_{n-2}, \ldots, x_1$ dalle equazioni da $n-1$ a 1 con le relazioni
$$x_i = \frac{a_{i,n+1}}{a_{ii}} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j$$

Ci sono molte varianti dello schema di eliminazione di Gauss. La sostituzione a ritroso può essere fatta eliminando gli elementi sopra la diagonale per mezzo di semplici combinazioni di righe e procedendo in su dall'ultima riga (come nel paragrafo precedente). Gli elementi diagonali possono tutti essere resi uguali ad 1 come primo passo prima della eliminazione degli elementi nella colonna corrispondente. Questo anticipa la divisione che normalmente si fa nella sostituzione a ritroso. Una variante del metodo di Gauss è il cosidetto metodo di *Gauss-Jordan* in cui l'eliminazione riguarda non solo gli elementi sotto la diagonale principale ma anche quelli sopra. In genere gli elementi diagonali vengono resi uguali ad 1 in modo da ottenere alla fine del processo di eliminazione la matrice unitaria e quindi il risultato direttamente come la colonna dei termini noti. Il pivoting è normalmente usato per preservare l'accuratezza numerica. Tuttavia, seppur dal punto di vista logico questo schema appare più naturale della semplice eliminazione di Gauss, dal punto di vista numerico richiede circa il doppio delle operazioni ed è per questo sconsigliato.

A volte si è interessati a trovare simultaneamente la soluzione di un dato sistema di equazioni per diverse realizzazioni del vettore termine noto. Allora si può procedere come fatto prima aumentando la matrice con tutti i diversi vettori noti $\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \ldots, \mathbf{b}^{(k)}$, procedendo all'eliminazione di Gauss e poi nella sostituzione a ritroso le diverse soluzioni saranno ottenute usando i trasformati dei vettori termini noti corrispondenti.

3.2.2 Scaling

A volte le righe della matrice aumentata debbono essere scalate prima di poter fare la scelta appropriata del "pivot". Lo **scaling** è l'operazione di aggiustamento dei coefficienti di un sistema di equazioni in modo che risultino tutti dello stesso ordine di grandezza. Se in un equazione ci sono coefficienti molto diversi tra loro e operiamo con il pivoting senza scaling rischiamo di usare come pivot un numero molto minore degli altri coefficienti della stessa riga, e questo può esaltare l'errore di arrotondamento invece di ridurlo, motivo per il quale si è operato con il pivoting. Consideriamo il seguente esempio

$$\left[\begin{array}{rrr|r} 3 & 2 & 100 & 105 \\ -1 & 3 & 100 & 102 \\ 1 & 2 & -1 & 2 \end{array}\right] \tag{3.25}$$

e usiamo solo 3 cifre significative. Usando pivoting parziale (solo righe) otterremmo

$$\left[\begin{array}{ccc|c} 3.00 & 2.00 & 100 & 105 \\ 0.00 & 3.66 & 133 & 137 \\ 0.00 & 0.00 & -82.6 & -82.7 \end{array}\right] \Longrightarrow \left\{\begin{array}{l} x_1 = 1.00 \\ x_2 = 1.09 \\ x_3 = 0.94 \end{array}\right. \tag{3.26}$$

da confrontare con la soluzione esatta ($x_1 = x_2 = x_3 = 1$). Se invece scaliamo i valori di ogni riga dividendo per il suo elemento più grande prima dell'eliminazione si ha

$$\left[\begin{array}{ccc|c} 0.03 & 0.02 & 1.00 & 1.05 \\ -0.01 & 0.03 & 1.00 & 1.02 \\ 0.50 & 1.00 & -0.50 & 1.00 \end{array}\right] \tag{3.27}$$

ed eliminando si ottiene

$$\left[\begin{array}{ccc|c} 0.50 & 1.00 & -0.50 & 1.00 \\ 0.00 & 0.05 & 0.99 & 1.04 \\ 0.00 & 0.00 & 1.82 & 1.82 \end{array}\right] \Longrightarrow \left\{\begin{array}{l} x_1 = 1.00 \\ x_2 = 1.00 \\ x_3 = 1.00 \end{array}\right. \tag{3.28}$$

che dà la soluzione identica a quella esatta nella precisione considerata.

Lo scaling è vantaggioso solo se gli elementi di una stessa riga sono molto diversi tra loro in modulo. Se sono tutti di stesso ordine di grandezza va evitato perchè costa e introduce ulteriori errori di arrotondamento.

3.3 Decomposizione LU

Il metodo dell'eliminazione può essere modificato in diversi modi per ottenere altre decomposizioni LU. Infatti ogni matrice A che ha tutti gli elementi sulla diagonale non nulli può essere decomposta nel prodotto di una matrice triangolare inferiore (L) e una tringolare superiore (U) in un infinità di modi. Si chiama **LU decomposition** o *Cholesky /Crout reduction* la particolare scelta di LU in modo che U abbia tutti gli elementi diagonali uguali a 1. Dalla condizione che $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$ si ottiene (nel caso 4×4)

$$\left[\begin{array}{cccc} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{array}\right] \cdot \left[\begin{array}{cccc} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \tag{3.29}$$

Moltipicando le righe di **L** per la prima colonna di **U** si ottiene

$$l_{11} = a_{11}; \quad l_{21} = a_{21}; \quad l_{31} = a_{31}; \quad l_{41} = a_{41} \tag{3.30}$$

ossia la prima colonna di **L** è uguale alla prima colonna di **A**. Moltiplicando la prima riga di **L** per le colonne di **U** si ottiene

$$u_{12} = \frac{a_{12}}{l_{11}}; \quad u_{13} = \frac{a_{13}}{l_{11}}; \quad u_{14} = \frac{a_{14}}{l_{11}} \tag{3.31}$$

Abbiamo quindi determinato la prima colonna di **U**. In questo modo alternando possiamo ottenere le colonne di **L** e le righe di **U** in successione. La forma generale delle equazioni per ottenere la decomposizione LU sono

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}, \qquad j \leq i, \qquad i = 1, 2, \ldots, n \tag{3.32}$$

$$u_{ij} = \frac{a_{ij}}{l_{ii}} - \frac{1}{l_{ii}} \sum_{k=1}^{i-1} l_{ik} u_{kj}, \qquad i \leq j, \qquad j = 2, 3, \ldots, n \tag{3.33}$$

Questo metodo è piuttosto popolare perchè lo spazio di memoria (storage) necessario può essere limitato osservando che non è necessario memorizzare gli zeri in **L** e **U** come anche la diagonale di U che è sempre 1. Inoltre dalle equazioni appena scritte si può vedere che una volta che l'elemento generico a_{ij} è stato usato nella procedura non è usato una seconda volta quindi la sua locazione di memoria può essere usata per un elemento di **L** o **U**.

Una volta che la matrice originale è stata decomposta, la soluzione del problema lineare originale, $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, è facilmente ottenibile per un qualsiasi vettore dei termini noti **b**. Gli elementi della matrice **L** non sono altro che una registrazione delle operazioni necessarie per trasformare la matrice **A** in **U**. Dobbiamo allora applicare le stesse trasformazioni a **b** per ottenere un nuovo vettore $\mathbf{b}'$. Se poi applichiamo la sostituzione a ritroso a $\mathbf{b}'$ tramite la matrice **U** otteniamo la soluzione. La equazioni per ottenere $\mathbf{b}'$ sono

$$b'_1 = \frac{b_1}{l_{11}} \tag{3.34}$$

$$b'_i = \frac{b_i}{l_{ii}} - \frac{1}{l_{ii}} \sum_{k=1}^{i-1} l_{ik} b'_k, \qquad j = 2, 3, \ldots, n \tag{3.35}$$

mentre le equazioni della back-substitution sono

$$x_n = b'_n \tag{3.36}$$

$$x_j = b'_j - \sum_{k=j+1}^{n} u_{jk} x_k, \qquad j = n-1, n-2, \ldots, 1 \tag{3.37}$$

Un vantaggio della decomposizione LU rispetto al metodo dell'eliminazione è che in LU possiamo usare la doppia precisione solo nel calcolare le somme e quindi risparmiare in memoria (è importante per matrici grandi) oppure a parità di memoria ottenere un'accuratezza maggiore che nell'uso della singola precisione usando semplicemente pochi numeri in doppia precisione. Inoltre il metodo LU può essere adattato per risolvere lo stesso problema per molti diversi vettori dei termini noti senza un grande sforzo supplementare. Il pivoting in combinazione con il metodo LU è più complicato che nell'eliminazione. Questo perchè non trattiamo il vettore dei temini noti contemporaneamente alla matrice dei coefficienti. Per usare il pivoting è necessario registrare tutte le operazioni di inversione delle righe della matrice dei coefficienti operate durante la decomposizione ad applicare le stesse inversioni ad ogni diverso vettore dei termini noti.

3.4 Matrici Singolari

Un set arbitrario di equazioni lineari può non avere soluzione unica, o non essere risolubile.

- Se il numero di equazioni m è minore del numero di incognite n non è possibile trovare una soluzione unica del sistema. In questo caso possiamo trovare invece una famiglia di soluzioni esprimendo m variabile in termini delle restanti $n - m$.

- Il caso opposto, ossia il numenro di equazioni m è maggiore del numero di incognite n è anch'esso patologico. Possiamo normalmente trovare un sottoinsieme di n equazioni che ammettono soluzione. Dopodichè dobbiamo distinguere i due casi:
  a) se le $m - n$ equazioni rimanenti sono soddisfatte dalla soluzione trovata, allora esiste soluzione unica del sistema e $m - n$ equazioni sono *ridondanti*
  b) se la soluzione delle prime n equazioni non soddisfa le rimanenti $m - n$ il sistema è *inconsistente*, e non ammette soluzione.

In generale se un sistema $n \times n$ non ammette soluzione unica, la matrice dei coefficienti si dice **singolare**. Se la matrice dei coefficienti può essere triangolarizzata senza avere zeri sulla diagonale la matrice si dice **non singolare**. Quando la dimensione n del problema diventa grande non è semplice dire a vista se una matrice è singolare (e quindi il problema è ridondante o inconsistente) oppure è non singolare (e quindi il problema ammette soluaione unica). Per sapere se un problema è singolare o meno bisogna calcolare il **rango** della matrice dei coefficienti, ossia il numero di righe o colonne che sono linearmente indipendenti. Per calcolare il rango si può triangolarizzare la matrice con l'eliminazione di Gauss. Se non appaiono zeri sulla diagonale della matrice triangolare allora la matrice originaria ha rango pari alla dimesione del problema $r = n$. In questo caso si dice che la matrice ha rango completo, e il problema ammette soluzione unica. Se, nonostante il pivoting, la forma triangolare ha uno o più zeri sulla diagonale allora il sistema non ammette soluzione unica. Tuttavia il sistema è consistente (e quindi ridondante) se nella procedura di sostituzione a ritroso si ottiene la forma indeterminata (0/0). Se invece si ottiene la divisione di un numero diverso da zero per zero allora il sistema è inconsistente. Consideriamo ad esempio la matrice aumentata

$$\left[\begin{array}{rrr|r} 1 & -2 & 3 & 5 \\ 2 & 4 & -1 & 7 \\ -1 & -14 & 11 & 2 \end{array}\right] \tag{3.38}$$

Con la riduzione di Gauss si ottiene

$$\left[\begin{array}{rrr|r} 1 & -2 & 3 & 5 \\ 0 & 8 & -7 & -3 \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{3.39}$$

che è chiaramente inconsistente. Se il terzo termine noto fosse stato 1 invece di 2 avremmo ottenuto

$$\left[\begin{array}{rrr|r} 1 & -2 & 3 & 5 \\ 0 & 8 & -7 & -3 \\ 0 & 0 & 0 & 0 \end{array}\right] \tag{3.40}$$

che invece è solo ridondante.

Come già detto il caso di matrici singolari richiede algoritmi diversi dai vari schemi di decomposizione visti in queste note.

3.5 Determinante e inversione di matrice

Determinante

Ci si può chiedere come mai non si è menzionato fin qui il metodo dei determinanti per risolvere i sistemi lineari (metodo di Cramer). Il motivo è che, con l'eccezione di sistemi di poche equazioni, il metodi di Cramer è troppo inefficiente. Ad esempio risolvere un sistema di 10 equazioni richiederebbe circa 70×10^6 moltiplicazioni e divisioni se si usa il metodo standard di espansione in termini dei minori. Con metodi più efficienti di calcolare i determinanti ci si può ridurre a circa 3000 operazioni, sempre molto rispetto alle circa 380 richieste del metodo di eliminazione di Gauss. Abbiamo visto che la matrice originale viene decomposta LU nel metodo di Gauss. Inoltre il determinante di una matrice triangolare è dato dal prodotto degli elementi sulla diagonale, e la matrice L ha determinante unitario essendo triangolare e avendo tutti elementi diagonali uguali a 1. Sapendo che $det(\mathbf{A} \cdot \mathbf{B}) = det(\mathbf{A})det(\mathbf{B})$, si ottiene che $det(\mathbf{A}') = det(\mathbf{U}) = \prod_i u_{ii}$. Inoltre ricordando che a) lo scambio di due righe o due colonne cambia il segno del determinante; b) la moltiplicazione di una riga (o colonna) per una costante moltiplica il determinante per la stessa costante, si può risalire dal determinante della forma

triangolarizzata al determinante originale tenendo conto in maniera opportuna delle trasformazioni operate nel corso dell'eliminazione. Ad esempio se si è usato solo il pivoting per ridurre la matrice allora si avrà

$$det(A) = (-)^m \prod_{i=1}^{n} u_{ii} \tag{3.41}$$

dove m è il numero di interscambi di righe operati per il pivoting.

Matrice inversa

Data una matrice $\mathbf{A}$ si definisce la sua matrice inversa $\mathbf{A}^{-1}$ quella matrice che moltiplicata per $\mathbf{A}$ fornisce la matrice identità

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{1} \tag{3.42}$$

La matrice inversa può essere ottenuta in termine dei minori del Det(A), ma non è una procedura utile in pratica. La tecnica di Gauss-Jordan può essere applicata per ottenere in pratica la matrice inversa $\mathbf{A}^{-1}$.

Si deve aumentare la matrice originale con la matrice identità e ridurre la matrice originale all'identità con operazioni elementari di riduzione, operando anche sulla parte destra della matrice aumentata. Quando la matrice identità compare al posto della matrice di partenza, allora a destra comparrà la matrice inversa. Esempio

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix} \tag{3.43}$$

Aumentiamo con l'identità ed eliminiamo la prima colonna

$$\left(\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array}\right) \rightarrow \left(\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 3 & -5 & -3 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right) \tag{3.44}$$

Scambiamo ora seconda e terza riga ed eliminiamo l'elemento 32

$$\left(\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 3 & -5 & -3 & 1 & 0 \end{array}\right) \rightarrow \left(\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -5 & 0 & 1 & -3 \end{array}\right) \tag{3.45}$$

dividiamo la terza riga per -5 ed eliminiamo tramite di essa l'elemento 13

$$\left(\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1/5 & 3/5 \end{array}\right) \rightarrow \left(\begin{array}{ccc|ccc} 1 & -1 & 0 & 1 & 2/5 & -6/5 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1/5 & 3/5 \end{array}\right) \tag{3.46}$$

eliminiamo ora l'elemento 12 sommando la riga 2 alla riga 1

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 2/5 & -1/5 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1/5 & 3/5 \end{array}\right) \Longrightarrow \mathbf{A}^{-1} = \begin{pmatrix} 0 & 2/5 & -1/5 \\ -1 & 0 & 1 \\ 0 & -1/5 & 3/5 \end{pmatrix} \tag{3.47}$$

In linea di principio, conoscere l'inversa ci permette anche di conoscere la soluzione del problema $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ perchè la soluzione è, per ogni termine noto, $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$. Tuttavia questo metodo di risoluzione non è efficiente rispetto alla decomposizione LU. Infatti il calcolo dell'inversa richiede di risolvere il problema con n vettori colonna di termini noti, mentre la decomposizione LU è solo la riduzione alla forma triangolare. Tuttavia usando il concetto di inversa si può capire

meglio la procedura di soluzione (eliminazione). Abbiamo visto che nella decomposizione LU si ha: $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$ quindi il problema originario è

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{L} \cdot \mathbf{U} \cdot \mathbf{x} = \mathbf{b} \tag{3.48}$$

moltiplicando a sinistra per $\mathbf{L}^{-1}$ si ottiene

$$\mathbf{L}^{-1} \cdot \mathbf{L} \cdot \mathbf{U} \cdot \mathbf{x} = \mathbf{L}^{-1} \cdot \mathbf{b} \tag{3.49}$$

Definendo $\mathbf{b} = \mathbf{L} \cdot \mathbf{b}'$ la soluzione del problema originale è equivalente alla soluzione dei due sistemi accoppiati

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{b}' \tag{3.50}$$
$$\mathbf{L} \cdot \mathbf{b}' = \mathbf{b} \tag{3.51}$$

Entrambi i sistemi sono semplici perché le matrici sono triangolari. Una volta operata la decomposizione LU risolveremo prima il secondo sistema ottenendo $\mathbf{b}'$ e poi il primo sistema con la backsubstitution per ottenere $\mathbf{x}$.

3.6 Norma

Dobbiamo ora estendere il concetto di norma, come misura della grandezza di un oggetto, alle matrici. Possiamo individuare questa misura di grandezza in molti modi diversi. Dobbiamo però richiedere che ogni definizione soddisfi quattro proprietà fondamentali

1. la norma deve essere sempre non negativa, essendo nulla solo per la matrice che ha tutti gli elementi nulli: $||\mathbf{A}|| \geq 0 \qquad ||\mathbf{A}|| = 0 \Leftrightarrow \mathbf{A} = \mathbf{0}$

2. La norma di una matrice moltiplicata per uno scalare è la norma della matrice per lo scalare: $||k\mathbf{A}|| = k||\mathbf{A}||$

3. La norma della somma di due matrici è sempre minore o uguale alla somma delle norme delle due matrici: $||\mathbf{A} + \mathbf{B}|| \leq ||\mathbf{A}|| + ||\mathbf{B}||$

4. La norma del prodotto righe per colonne di due matrici è sempre non maggiore del prodotto delle norme delle due matrici: $||\mathbf{A} \cdot \mathbf{B}|| \leq ||\mathbf{A}||\ ||\mathbf{B}||$

Notiamo che la terza condizione è la nota disuguaglianza triangolare o di Schwartz.

Consideriamo dapprima il caso dei vettori ordinari in un spazio finito dimensionale di dimensione n. Si definisce *p-norma* di un vettore la relazione

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p \right)^{1/p} \qquad p = 1, \ldots, \infty \tag{3.52}$$

Alcuni casi particolari e più comuni sono

$$||\mathbf{x}||_1 = \sum_i |x_i| \qquad \text{somma dei moduli degli elementi} \tag{3.53}$$
$$||\mathbf{x}||_2 = \left(\sum_i |x_i|^2 \right)^{1/2} \qquad \text{norma euclidea= modulo quadrato} \tag{3.54}$$
$$||\mathbf{x}||_\infty = \max_{1 \leq i \leq n} |x_i| \qquad \text{massimo elemento} \tag{3.55}$$

Passiamo ora alle matrici. Per estensione diretta possiamo definire

$$||\mathbf{A}||_1 = \max_{1\leq j\leq n}\left[\sum_{i=1}^{n}|a_{ij}|\right] \qquad \text{colonna con massima somma} \tag{3.56}$$

$$||\mathbf{A}||_\infty = \max_{1\leq i\leq n}\left[\sum_{j=1}^{n}|a_{ij}|\right] \qquad \text{riga con massima somma} \tag{3.57}$$

$||\mathbf{A}||_2$ invece non è facilmente calcolabile. Si può dimostrare che è collegata agli autovalori della matrice $\mathbf{A}$. Si può anche dimostrare che $||\mathbf{A}||_2$ assume il valore minore tra tutte le possibili norme. In alternativa possiamo utilizzare la norma *euclidea* che nel caso dei vettori coincide con la norma di ordine 2. La norma euclidea è definita da

$$||\mathbf{A}||_e = \left[\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}|^2\right]^{1/2} \tag{3.58}$$

Il concetto di norma è importante per quantificare, ad esempio, la norma del vettore differenza tra la soluzione numerica e quella esatta di un problema, ossia per quantificare l'accuratezza di ogni soluzione numerica.

3.7 Condition number e stima dell'errore sulla soluzione

Risolvendo numericamente un problema lineare $\mathbf{A}\cdot\mathbf{x}=\mathbf{b}$, speriamo che la soluzione ottenuta sia vicina a quella esatta. Pivoting e miglioramento iterativo (vedi dopo) possono aumentare l'accuratezza numerica delle soluzioni. Tuttavia, anche in questo caso, alcuni sistemi possono essere estremamente sensibili all'errore di round-off e la soluzione essere piuttosto inaccurata. In questi casi il sistema è detto **instabile** e la matrice $\mathbf{A}$ è detta **ill-conditioned** (malamente condizionata). Una misura di questa instabilità è data dal **condition number**. Consideriamo un esempio di un sistema instabile e consideriamo la soluzione che si ottiene per piccole variazioni del termine noto. Consideriamo il problema:

$$\begin{pmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2.00 \\ 2.00 \end{pmatrix} \tag{3.59}$$

la cui soluzione esatte è $(1.00, 1.00)$. Supponiamo adesso di perturbare di solo 2 unità sull'ultima cifra significativa il termine noto in maniera che la somma delle componenti rimanga uguale e vediamo come cambia la soluzione:

$$\begin{pmatrix} 2.00 \\ 2.00 \end{pmatrix} \Rightarrow \begin{pmatrix} 2.02 \\ 1.98 \end{pmatrix} \rightarrow \begin{pmatrix} x = 2.00 \\ y = 0.00 \end{pmatrix} \tag{3.60}$$

$$\begin{pmatrix} 2.00 \\ 2.00 \end{pmatrix} \Rightarrow \begin{pmatrix} 1.98 \\ 2.02 \end{pmatrix} \rightarrow \begin{pmatrix} x = 0.00 \\ y = 2.00 \end{pmatrix} \tag{3.61}$$

quindi vediamo una forte sensibilità a piccole variazioni dei coefficienti.

Vediamo un altro esempio:

$$\begin{pmatrix} 3.02 & -1.05 & 2.53 \\ 4.33 & 0.56 & -1.78 \\ -0.83 & -0.54 & 1.47 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1.61 \\ 7.23 \\ -3.38 \end{pmatrix} \tag{3.62}$$

La soluzione esatta del sistema è

$$\mathbf{x} = \begin{pmatrix} 1.00 \\ 2.00 \\ -1.00 \end{pmatrix} \tag{3.63}$$

Risolvendo il sistema con l'eliminazione Gaussiana con pivoting e con 3 cifre significative (con arrotondamento) il sistema aumentato e triangolarizzato risulta

$$\left(\begin{array}{ccc|c} 4.33 & 0.56 & -1.78 & 7.23 \\ 0.00 & -1.44 & 3.77 & -6.65 \\ 0.00 & 0.00 & -0.00362 & 0.00962 \end{array}\right) \tag{3.64}$$

che fornisce la soluzione (per backsubstitution)

$$\tilde{\mathbf{x}} = \begin{pmatrix} 0.88 \\ -2.35 \\ -2.66 \end{pmatrix} \tag{3.65}$$

molto diversa da quella esatta. Vediamo quindi come ci sia di nuovo una forte sensibilità all'errore di round-off. Se avessimo operato con 6 cifre significative avremmo ottenuto una precisione molto migliore. In questo esempio la ill-conditioness della matrice dei coefficienti si rispecchia nell'elemento diagonale molto piccolo della forma triangolarizzata. Questo significa che $\det(\mathbf{U}) = \det(\mathbf{A}) \simeq 0$ e quindi la matrice è quasi singolare.

Un altro aspetto della ill-conditioness è la sensibilità a piccole variazioni degli elementi della matrice dei coefficienti $\mathbf{A}$. Se in questo esempio avessimo cambiato a_{11} da 3.02 a 3.00 avremmo ottenuto una soluzione (esatta)

$$\mathbf{x} = \begin{pmatrix} 1.07968 \\ 4.75637 \\ 0.05856 \end{pmatrix} \tag{3.66}$$

Quindi in un sistema instabile la soluzione è estremamente sensibile a piccole variazioni dei coefficienti (sia della matrice che del termine noto).

L'altra faccia della stessa medaglia è che in un sistema ill-conditioned non è possibile sapere l'accuratezza della soluzione ottenuta semplicemente sostituendola nel sistema per verificare se si riottiene il termine noto. Questo perché il sistema è pochissimo sensibile a variazioni della soluzione $\mathbf{x}$. Consideriamo l'esempio di prima. Se usiamo la soluzione esatta

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \Rightarrow \mathbf{b} = \mathbf{A} \cdot \mathbf{x} = \begin{pmatrix} -1.61 \\ 7.23 \\ -3.38 \end{pmatrix} \tag{3.67}$$

se invece sostituiamo la soluzione approssimata

$$\tilde{\mathbf{x}} = \begin{pmatrix} 0.88 \\ -2.35 \\ -2.66 \end{pmatrix} \Rightarrow \tilde{\mathbf{b}} = \mathbf{A} \cdot \tilde{\mathbf{x}} = \begin{pmatrix} -1.6047 \\ 7.2292 \\ -3.3716 \end{pmatrix} \tag{3.68}$$

che è comunque molto viscino a $\mathbf{b}$.

Definiamo **errore** di una soluzione $\tilde{\mathbf{x}}$ la differenza di questa soluzione con la soluzione esatta (ma incognita)

$$\mathbf{e}(\tilde{\mathbf{x}}) = \mathbf{x} - \tilde{\mathbf{x}} \tag{3.69}$$

e definiamo **residuo** di una soluzione $\tilde{\mathbf{x}}$ la differenza tra il vettore dei termini noti $\mathbf{b}$ e il vettore $\tilde{\mathbf{b}} = \mathbf{A} \cdot \tilde{\mathbf{x}}$

$$\mathbf{r}(\tilde{\mathbf{x}}) = \mathbf{b} - \mathbf{A} \cdot \tilde{\mathbf{x}} \tag{3.70}$$

L'esempio appena presentato mostra che la norma del residuo, $||\mathbf{r}||$, non è una buona misura della norma dell'errore, $||\mathbf{e}||$, per un sistema ill-conditioned. Cerchiamo quindi una misura dell'errore commesso nel risolvere un sistema lineare. Questa misura è data dal **condition number** di una matrice. Il **condition number** è definito da

$$C(\mathbf{A}) = ||\mathbf{A}||\ ||\mathbf{A}^{-1}|| \geq 1 \qquad\qquad C(\mathbf{1}) = 1 \tag{3.71}$$

Il condition number ci permette di legare la norma dell'errore alla norma del residuo. Infatti possiamo scrivere

$$\mathbf{r}(\tilde{\mathbf{x}}) = \mathbf{b} - \mathbf{A} \cdot \tilde{\mathbf{x}} = \mathbf{A} \cdot (\mathbf{x} - \tilde{\mathbf{x}}) = \mathbf{A} \cdot \mathbf{e}(\tilde{\mathbf{x}}) \tag{3.72}$$

da cui segue che

$$\mathbf{e}(\tilde{\mathbf{x}}) = \mathbf{A}^{-1} \cdot \mathbf{r}(\tilde{\mathbf{x}}) \tag{3.73}$$

Dalla definizione di norma sappiamo che

$$||\mathbf{e}|| \leq ||\mathbf{A}^{-1}||\ ||\mathbf{r}|| \tag{3.74}$$

e anche che

$$||\mathbf{r}|| \leq ||\mathbf{A}||\ ||\mathbf{e}|| \tag{3.75}$$

utilizzando queste due diseguaglianze possiamo ottenere

$$\frac{||\mathbf{r}||}{||\mathbf{A}||} \leq ||\mathbf{e}|| \leq ||\mathbf{A}^{-1}||\ ||\mathbf{r}|| \tag{3.76}$$

Applicando lo stesso ragionamento al sistema originario $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ e al suo inverso $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$ si ottiene

$$\frac{||\mathbf{b}||}{||\mathbf{A}||} \leq ||\mathbf{x}|| \leq ||\mathbf{A}^{-1}||\ ||\mathbf{b}|| \tag{3.77}$$

Combinando queste due relazioni si ottiene

$$\frac{1}{C(A)} \frac{||\mathbf{r}||}{||\mathbf{b}||} \leq \frac{||\mathbf{e}||}{||\mathbf{x}||} \leq C(A) \frac{||\mathbf{r}||}{||\mathbf{b}||} \tag{3.78}$$

con

$$\frac{||\mathbf{r}||}{||\mathbf{b}||} = \text{residual relativo} \tag{3.79}$$

$$\frac{||\mathbf{e}||}{||\mathbf{x}||} = \text{errore relativo} \tag{3.80}$$

La relazione (**??**) ci dice che l'errore relativo è sempre non superiore al residual relativo moltiplicato per il condition number. Quindi:

1. se $C(A) \gg 1$ il residual relativo non dà informazioni sull'errore relativo e ci si deve preoccupare della qualità della soluzione trovata
2. quando $C(A) \sim 1$ allora il residual relativo è una buona stima dell'errore relativo sulla soluzione

Se la matrice è ill-conditioned abbiamo $C(A) \gg 1$. Infatti gli elementi di $\mathbf{A}^{-1}$ saranno grandi rispetto a quelli di $\mathbf{A}$. Quindi la norma $||\mathbf{A}^{-1}||$ potrebbe fornire già indicazioni sulla ill-conditioning. Tuttavia possiamo avere casi in cui gli elementi di $\mathbf{A}$ sono piccoli senza avere condizioni di ill-conditioning. Moltiplicare $||\mathbf{A}^{-1}||$ per $||\mathbf{A}||$ fornisce in questi casi un condition number $C(A)$ vicino a 1, mentre non lo fa per matrici ill-conditioned.

Il condition number può anche essere usato per stimare la propagazione di eventuali errori nei coefficienti di $\mathbf{A}$. Consideriamo il caso in cui i coefficienti di A sono il risultato di misure e quindi affetti da errori sperimentali. Se $\mathbf{x}$ è la soluzione di $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ e $\tilde{\mathbf{x}}$ la soluzione di $(\mathbf{A} + \mathbf{E}) \cdot \tilde{\mathbf{x}} = \tilde{\mathbf{A}} \cdot \tilde{\mathbf{x}} = \mathbf{b}$ dove $\mathbf{E}$ rappresenta la matrice degli errori, possiamo sapere quanto è grande l'errore sulla soluzione $\mathbf{e} = \mathbf{x} - \tilde{\mathbf{x}}$

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b} = \mathbf{A}^{-1} \cdot (\tilde{\mathbf{A}} \cdot \tilde{\mathbf{x}}) = \mathbf{A}^{-1} \cdot (\mathbf{A} + \tilde{\mathbf{A}} - \mathbf{A}) \cdot \tilde{\mathbf{x}} = \tilde{\mathbf{x}} + \mathbf{A}^{-1} \cdot (\tilde{\mathbf{A}} - \mathbf{A}) \cdot \tilde{\mathbf{x}} \tag{3.81}$$

da cui

$$\mathbf{x} - \tilde{\mathbf{x}} = \mathbf{A}^{-1} \cdot \mathbf{E} \cdot \tilde{\mathbf{x}} \tag{3.82}$$

e quindi

$$||\mathbf{x} - \tilde{\mathbf{x}}|| \leq ||\mathbf{A}^{-1}||\ ||\mathbf{E}||\ ||\tilde{\mathbf{x}}|| = ||\mathbf{A}^{-1}||\ ||\mathbf{A}||\ \frac{||\mathbf{E}||}{||\mathbf{A}||}\ ||\tilde{\mathbf{x}}|| \tag{3.83}$$

L'errore relativo obbedisce quindi alla diseguaglianza

$$\frac{||\mathbf{x} - \tilde{\mathbf{x}}||}{||\tilde{\mathbf{x}}||} \leq C(A) \frac{||\mathbf{E}||}{||\mathbf{A}||} \tag{3.84}$$

Il condition number quindi regola anche la propagazione sulla soluzione numerica di errori commessi sui coefficienti della matrice $\mathbf{A}$.

3.8 Miglioramento iterativo della soluzione

Qualunque metodo numerico fornisce, a causa dell'errore di round-off, una soluzione approssimata $\tilde{\mathbf{x}}$ del problema $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. E' sempre possibile migliorare la qualità di questa soluzione con un metodo iterativo. Consideriamo il vettore errore $e = x - \tilde{x}$ e il vettore residual $r = b - A \cdot \tilde{x}$ e la loro relazione già vista prima $A \cdot e = r$. Possiamo ora risolvere questo problema ed ottenere una stima di $\mathbf{e}$ da aggiungere alla soluzione $\tilde{\mathbf{x}}$ precedentemente ottenuta per migliorare la stime di $\mathbf{x}$. Tuttavia bisogna in genere calcolare il residual con il massimo di precisione possibile (doppia) altrimenti rischiamo di non migliorare ma di peggiorare la soluzione. Vediamo in dettaglio come si procede in questa soluzione iterativa. Consideriamo la decomposizione LU di A. In realtà, nonostante il pivoting e lo scaling, il prodotto $\mathbf{L} \cdot \mathbf{U}$ non fornirà esattamente la matrice di partenza $\mathbf{A}$. Chiamiamo $\mathbf{B}_0$ una prima approssimazione di $\mathbf{A}^{-1}$ che si ottiene con la decomposizione LU. Definiamo la matrice resto come

$$\mathbf{R} = \mathbf{1} - \mathbf{B}_0 \cdot \mathbf{A} \tag{3.85}$$

infatti se $\mathbf{B}_0 = \mathbf{A}^{-1}$ allora $\mathbf{R} = \mathbf{0}$. Abbiamo quindi $\mathbf{B}_0 \cdot \mathbf{A} = \mathbf{1} - \mathbf{R}$. Scriviamo ora

$$\mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot (\mathbf{B}_0^{-1} \cdot \mathbf{B}_0) = (\mathbf{A}^{-1} \cdot \mathbf{B}_0^{-1}) \cdot \mathbf{B}_0 = (\mathbf{B}_0 \cdot \mathbf{A})^{-1} \cdot \mathbf{B}_0 = (\mathbf{1} - \mathbf{R})^{-1} \cdot \mathbf{B}_0 \tag{3.86}$$

Sapendo che una qualunque funzione di matrice è definita attraverso il suo sviluppo in serie di potenze (dove converge) possiamo scrivere

$$\mathbf{A}^{-1} = (\mathbf{1} - \mathbf{R})^{-1} \cdot \mathbf{B}_0 = \left(\sum_{k=0}^{\infty} \mathbf{R}^k \right) \cdot \mathbf{B}_0 \tag{3.87}$$

Chiamiamo $\mathbf{B}_n = \left(\sum_{k=0}^{n} \mathbf{R}^k \right) \cdot \mathbf{B}_0$, allora se il limite per $n \to \infty$ esiste si ha

$$\lim_{n \to \infty} \mathbf{B}_n = \mathbf{A}^{-1} \tag{3.88}$$

Definiamo ora $\mathbf{x}_n = \mathbf{B}_n \cdot \mathbf{b}$. Se il limite suddetto esiste si avrà

$$\lim_{n \to \infty} \mathbf{x}_n = \lim_{n \to \infty} \mathbf{B}_n \cdot \mathbf{b} = \mathbf{A}^{-1} \cdot \mathbf{b} = \mathbf{x} \tag{3.89}$$

Inoltre

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{B}_{n+1} \cdot \mathbf{b} = (\mathbf{1} + \mathbf{R} + \mathbf{R}^2 + \ldots + \mathbf{R}^{n+1}) \cdot \mathbf{B}_0 \cdot \mathbf{b} \\ &= \mathbf{B}_0 \cdot \mathbf{b} + \mathbf{R} \cdot (\mathbf{1} + \mathbf{R} + \ldots + \mathbf{R}^n) \cdot \mathbf{B}_0 \cdot \mathbf{b} = \mathbf{B}_0 \cdot \mathbf{b} + \mathbf{R} \cdot \mathbf{x}_n \end{aligned} \tag{3.90}$$

Ricordando la definizione di $\mathbf{R}$: $\mathbf{R} = \mathbf{1} - \mathbf{B}_0 \cdot \mathbf{A}$ si ottiene lo schema iterativo voluto

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{B}_0 \cdot (\mathbf{b} - \mathbf{A} \cdot \mathbf{x}_n) \tag{3.91}$$

3.9 Considerazioni numeriche: matrici e uso della memoria fisica

Per convenzione una matrice $n \times m$ è un oggetto a due indici, il primo indice indica la riga il secondo la colonna

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & \ldots & a_{2m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & \ldots & \ldots & \ldots & a_{nm} \end{pmatrix} \tag{3.92}$$

In un programma possiamo ugualmente definire matrici

```
A(n,m)
```

e indirizzare la memoria per mezzo di due indici come si fa simbolicamente. Tuttavia la memoria fisica nel computer individuata sequenzialmente. Quindi un array a più indici sarà sempre mappato su una sequenza unidimensionale nella memoria fisica del computer. Per un oggetto a due indici, come la matrici che abbiamo considarato in questo capitolo, l'immagazzinamento nella memoria si può fare in due diversi modi:

- *immagazzinamento per colonne*:
  $a_{11}, a_{21}, a_{31}, \ldots, a_{n1}, a_{12}, \ldots, a_{n2}, a_{13}, \ldots, a_{n3}, \ldots, \ldots, a_{1m}, \ldots, a_{nm}$
- *immagazzinamento per righe*:
  $a_{11}, a_{12}, a_{13}, \ldots, a_{1m}, a_{21}, \ldots, a_{2m}, a_{31}, \ldots, a_{3m}, \ldots, \ldots, a_{n1}, \ldots, a_{nm}$

Il linguaggio Fortran usa l'immagazzinamento per colonne, mentre il c usa l'immagazzinamento per righe. Notare che nell'immagazzinamento per colonne l'indice di riga è quello che cambia più velocemente, nell'altro caso è il contrario. In generale non è necessario occuparsi di questo dettaglio però bisogna capire la differenza tra dimensione **fisica** DF e dimensione **logica** DL di un vettore e di una matrice. La dimensione fisica è la dimenione dichiarata nella statement dichiarativo della variabile ed indica la massima dimensione utilizzabile per la variabile in esame. La dimensione logica indica la dimensione effetivamente in uso durante una particolare esecuzione del programma. Naturalmente deve sempre valere $DF \geq DL$. Nel caso di array unidimensionali non ci son problemi. Bisogna invece fare attenzione all'indirizzamento della memoria quando si usano array a più di una dimensione in connessione con la chiamata a subroutines. Bisogna rendersi conto del fatto che non appena si lavora con array a dimensione ≥ 2 e la dimensione logica è inferiore alla dimensione fisica, gli elementi dell'array non vengono immagazzinati sequenzialmente nella memoria ma sono sparsi. Facciamo l'esempio di un array a due indici con dimensione fisica 5×5 e dimensione logica 3×2. Nello statement dichiarativo avrò scritto ad esempio

```
.......
real*8 a(5,5)
.......
```

ma la matrice su cui effettivamente lavoro è

$$\begin{pmatrix} a_{11} & a_{12} & - & - & - \\ a_{21} & a_{22} & - & - & - \\ a_{31} & a_{32} & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \end{pmatrix} \tag{3.93}$$

che viene imagazzinata nella memoria (usiamo ad esempio il metodo per colonne del fortran) come
$a_{11}, a_{21}, a_{31}, -, -, a_{12}, a_{22}, a_{32}, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -,$
dove il segno $-$ indica che l'elemento corrispondente non è inizializzato e non deve essere usato nel calcolo. Un possibile problema sorge quando passo la matrice ad una subroutine attraverso lo statement di chiamata. Infatti in fortran quando specifico una variabile come argomento di chiamata in realtà sto solo specificando l'indirizzo di memoria del suo primo elemento (oppure dell'elemento specificato). Quindi chiamando una routine, se per errore non specifico sia la dimensione logica che fisica dell'array posso commettere un errore grossolano di memoria facendo lavorare la routine su una parte della memoria che non corrisponde a quella voluta. Questo tipo di errore non è naturalmente individuato dal compilatore ed è spesso molto difficile da eliminare. Torniamo al caso della nostra matrice di DF 5×5 e DL 3×2. La struttura

```
program matrice
real a(5,5)
.......
.......
call invert(a,3,2)
.......
.......
.......
.......
end

subroutine invert(a,n,m)
integer n,m
real a(n,m)
.......
.......
return
end
```

è sbagliata perché viene pasato l'indirizzo del primo elemento a_{11} della matrice a e nella routine vengono riservati 6 locazioni di memoria sequenziali a partire dall'indirizzo di a_{11}. L'immagine della matrice a nella routine invert è quindi
$a_{11}, a_{21}, a_{31}, -, -, a_{12}$
che non è la matrice originaria nel programma principale. La forma corretta della struttura è invece

```
program matrice
```

```
real a(5,5)
.......
.......
call invert(a,3,2,5)
.......
.......
.......
.......
end

subroutine invert(a,n,m,mp)
integer n,m,mp
real a(mp,mp)
.......
.......
return
end
```

in cui si passa anche la dimensione fisica e la matrice all'interno della routine è correttamente dimensionata alla sua dimensione fisica. In questo modo le immagini della memoria associata ad a nel programma principale e nella routine sono identiche.

3.10 Metodi iterativi per la soluzione di sistemi lineari (e non lineari) di equazioni

In contrasto ai metodi diretti di soluzione di un sistema di equazioni lineari come quelli visti nel corso del capitolo, possiamo ottenere una soluzione del sistema con metodi iterativi. Il vantaggio dei metodi iterativi è la possibilità di estenderli a sistemi non lineari.

3.10.1 Metodo di Jacobi

Consideriamo il solito sistema lineare $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. Come esempio pratico consideriamo il sistema di equazioni

$$\begin{aligned} 8x_1 + x_2 - x_3 &= 8 \\ 2x_1 + x_2 + 9x_3 &= 12 \\ x_1 - 7x_2 + 2x_3 &= -4 \end{aligned}$$

che ha soluzione analitica $\mathbf{x}^T = \{1, 1, 1\}$. Risolviamo ogni equazione per una variabile diversa scegliendo, se possibile, l'equazione in cui la variabile compare con coefficiente più grande:

$$\begin{aligned} 1. \quad & x_1 = 1 - \frac{x_2}{8} + \frac{x_3}{8} \\ 2. \quad & x_3 = \frac{12}{9} - \frac{x_1}{12} - \frac{x_2}{6} \\ 3. \quad & x_2 = \frac{4}{7} + \frac{x_1}{7} + \frac{2x_3}{7} \end{aligned}$$

Scegliamo una condizione iniziale $\mathbf{x}^{(0)}$ e sostituiamola al membro di destra di ogni equazione. Otteniamo in questo modo la predizione $\mathbf{x}^{(1)}$ della soluzione dopo una iterazione dello schema. In formule la procedura è la seguente

$$\mathbf{x}^{(n+1)} = \mathbf{G} \cdot \mathbf{x}^{(n)} = \mathbf{b}' - \mathbf{B} \cdot \mathbf{x}^{(n)} \tag{3.94}$$

dove $\mathbf{G}$ e $\mathbf{B}$ sono matrici quadrate e $\mathbf{b}'$ è il vettore dei termini noti ridotto. Per ottenere la relazione tra $\mathbf{G}$ e $\mathbf{B}$ e i dati originali del problema, $\mathbf{A}$ e $\mathbf{b}$, decomponiamo $\mathbf{A}$ nel seguente modo

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \tag{3.95}$$

dove $\mathbf{L}$ e $\mathbf{U}$ sono rispettivamente due matrici triangolari inferiori e superiori con gli elementi della matrice originale $\mathbf{A}$ e con diagonale principale nulla, mentre $\mathbf{D}$ è la matrice diagonale che ha come elementi la diagonale principale di $\mathbf{A}$. Il sistema si può quindi riscrivere

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} + \mathbf{U}) \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{x} = \mathbf{b} \tag{3.96}$$

ossia

$$\mathbf{x} = \mathbf{D}^{-1} \cdot \mathbf{b} - \mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U}) \cdot \mathbf{x} \tag{3.97}$$

da cui otteniamo le seguenti relazioni

$$\mathbf{b}' = \mathbf{D}^{-1} \cdot \mathbf{b} \tag{3.98}$$

$$\mathbf{B} = \mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U}) \tag{3.99}$$

Riassumendo il metodo di Jacobi consiste in

1. riarrangiare le righe della matrice originaria in modo che gli elementi sulla diagonale abbiano valore massimo rispetto agli altri elementi della stessa riga
2. scegliere un'approssimazione iniziale $\mathbf{x}^{(0)}$ per la soluzione e iterare con le seguenti equazioni

$$x_i^{(n+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1,\neq i}^{N} \frac{a_{ij}}{a_{ii}} x_j^{(n)} \qquad n = 1, 2, \ldots \tag{3.100}$$

3. condizione sufficiente per la convergenza dello schema è

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \qquad \forall i \in [1, N] \tag{3.101}$$

   se questa condizione è verificata la convergenza non dipende dalla condizione iniziale

3.10.2 Metodo di Gauss-Seidel

Una variante del metodo di Jacobi che rende più rapida la convergenza dello schema è il cosiddetto metodo di Gauss-Seidel. In questo metodo si procede come in Jacobi ma invece di itarerare con le equazioni **??** si usano le seguenti equazioni

$$x_i^{(n+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(n+1)} - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_j^{(n)} \qquad n = 1, 2, \ldots \tag{3.102}$$

in cui per ottenere la (n+1)-esima predizione della variabile x_j si usa la predizione n-esima delle variabile con indice $k > j$ e la predizione (n+1)-esima delle variabile con indice $k < j$. La condizione sufficiente per la convergenza rimane la stessa di quella del metodo di Jacobi. In forma matriciale il metodo di Gauss-Seidel si scrive

$$\mathbf{x}^{(n+1)} = \mathbf{D}^{-1} \cdot \mathbf{b} - \mathbf{D}^{-1} \cdot \mathbf{L} \cdot \mathbf{x}^{(n+1)} - \mathbf{D}^{-1} \cdot \mathbf{U} \cdot \mathbf{x}^{(n)} \tag{3.103}$$

Come detto il vantaggio è nella velocità di convergenza e nella stabilità dello schema. Tuttavia mentre il metodo di Jacobi è intrinsecamente parallelo, questa caratteritica è persa nel metodo di Gauss-Seidel.

3.10.3 Sistemi non lineari

Il problema di trovare le soluzioni di un sistema di equazioni non lineari è molto più difficile del problema di risolvere un sistema lineare. Non è facile capire se e quante soluzioni il sistema ammette soluzioni, e in generale le soluzioni possibili non è detto che siano reali. Comunque quando sappiamo che soluzion reali esistono possiamo provare ad estendere gli schemi iterativi appena visti al caso di sistemi non lineari.

Dato il sistema di equazioni non lineari

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

possiamo sempre riscriverlo in modo da risolvere ogni equazioni per una diversa variabile

$$\begin{aligned} x_1 &= g_1(x_1, x_2, \ldots, x_N) \\ x_2 &= g_2(x_1, x_2, \ldots, x_N) \\ \ldots & \quad \ldots\ldots\ldots \\ x_N &= g_N(x_1, x_2, \ldots, x_N) \end{aligned}$$

e provare a risolvere il sistema con il metodo iterativo. Consideriamo l'esempio del sistema

$$\begin{aligned} x^2 + y^2 &= 4 \\ e^x + y &= 1 \end{aligned}$$

che ha due soluzioni reali, le intersezioni tra il cerchio di raggio 2 centrato nell'origine e la curva $y = 1 - e^x$ come mostrato in figure **??**

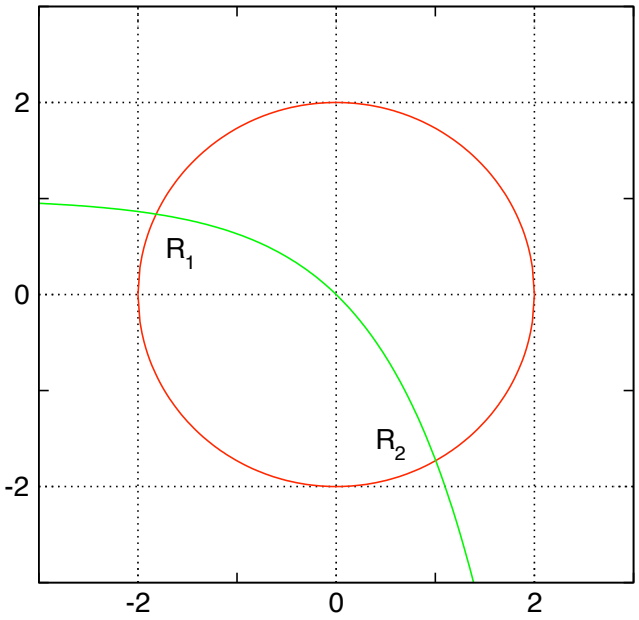

Figura 3.1: grafico delle equazioni $x^2 + y^2 = 4$ e $y = 1 - e^x$. Le due radici del sistema sono le intersezioni delle curve.

Capitolo 4

Derivazione e integrazione di funzioni unidimensionali

4.1 Derivazione numerica

Il problema che vogliamo discutere è: data una funzione nota per punti, possiamo calcolare le sue derivate in un punto?
È un problema di fondamentale importanza soprattutto in congiunzione con le equazioni differenziali, argomento centrale della fisica (anche numerica). Deriveremo almeno due forme diverse per stimare la derivata e ne calcoleremo gli errori. Come sempre è importante capire come gli errori numerici influenzano il risultato e che valore dei parametri è ottimale per una data lunghezza di parola.
Il punto di partenza è lo sviluppo di Taylor della funzione in esame $f(x)$ intorno al punto x di cui si vuole sapere la derivata

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{3!} + O(h^4) \tag{4.1}$$

Se ci accontentiamo di fermarci all'ordine 2 sappiamo che

$$f(x+h) = f(x) + f'(x)h + f''(\xi)\frac{h^2}{2} \qquad \xi \in [x, x+h] \tag{4.2}$$

Da questa espressione possiamo ottenere la **formula ad un punto** per la derivata prima

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi)}{2}h = \frac{f(x+h) - f(x)}{h} + O(h) \tag{4.3}$$

che ha una precisione di ordine h. Sappiamo che nel limite $h \to 0$ questa espressione tende alla vera derivata prima. Tuttavia in questo contesto il problema è capire quanto piccolo (o grande) deve essere l'ampiezza dell'intervallo h per avere una stima ottimale della derivata prima.

Uno schema più accurato si ottiene usando gli sviluppi per h e per $-h$

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2}f''(x) \pm \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(iv)}(x) + O(h^5) \tag{4.4}$$

Sottraendo i due sviluppi tra loro, tutti i termini pari scompaiono e rimane

$$\begin{aligned} f(x+h) - f(x-h) &= 2hf'(x) + \frac{h^3}{3}f'''(x) + O(h^5) \\ &= 2hf'(x) + \frac{h^3}{6}\left[f'''(\xi_1) + f'''(\xi_2)\right] \qquad \xi_1 \in [x, x+h]; \xi_2 \in [x-h, x] \end{aligned} \tag{4.5}$$

Ne segue lo **schema a due punti o della differenza centrale** per la derivata prima

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{4.6}$$

che ha una precisione di ordine h^2.

4.1.1 Analisi degli errori

In entrambi i metodi dobbiamo preoccuparci di due diversi tipi di errore: errore di troncamento ed errore di round-off.
L'errore di troncamento è quello stimato prima:

- $h\frac{f''(\xi)}{2}$ nello schema ad un punto
- $h^2\frac{f'''(\xi_1)+f'''(\xi_2)}{6}$ nello schema a due punti

L'errore di round-off invece viene dal fatto che in entrambi gli schemi, al numeratore compare la differenza di due numeri circa uguali cosa che fa perdere precisione. Il primo tipo di errore (troncamento) diminuisce al diminuire di h, il secondo (round-off) invece aumenta al diminuire di h. Bisogna allora porsi la domanda: che valore di h bisogna scegliere? Se escludiamo i casi banali di funzioni lineari e quadratiche (per le quali l'errore di troncamento è nullo), un criterio ragionevole è scegliere l'incremento h in maniera da minimizzare l'errore totale:

$$\bar{h} : \epsilon_{tot}(\bar{h}) = \{\epsilon_{tr}(\bar{h}) + \epsilon_{ro}(\bar{h})\} = min_h \tag{4.7}$$

In genere questa condizione si ha per $\epsilon_{tr} \simeq \epsilon_{ro}$.
Chiamiamo ϵ_m la precisione della macchina ($\epsilon_m \sim 10^{-7}$ in singola precisione, $\epsilon_m \sim 10^{-14}$ in doppia precisione). L'errore di round-off ϵ_{ro} è legato essenzialmente alla precisione

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h} \simeq \frac{\epsilon_m}{h} \Longrightarrow \epsilon_{ro} \simeq \frac{\epsilon_m}{h} \tag{4.8}$$

Lo stessa espressione si ottiene anche per lo schema alle differenze centrali. Invece l'errore di troncamento dipende dallo schema scelto

$$\begin{aligned} \epsilon_{tr} &\simeq \frac{|f''|}{2}h && \text{schema ad 1 punto} \\ \epsilon_{tr} &\simeq \frac{|f'''|}{6}h^2 && \text{schema alle differenze centrali} \end{aligned} \tag{4.9}$$

Imponendo l'uguaglianza dei due errori $\epsilon_{ro} \simeq \epsilon_{tr}$ si ottiene

$$\begin{aligned} \frac{\epsilon_m}{h} \simeq \frac{|f''|}{2}h \Longrightarrow \bar{h}_{1p}^2 \simeq \frac{2\epsilon_m}{|f''|} && \text{schema ad 1 punto} \\ \frac{\epsilon_m}{h} \simeq \frac{|f'''|}{6}h^2 \Longrightarrow \bar{h}_{dc}^3 \simeq \frac{6\epsilon_m}{|f'''|} && \text{differenze centrale} \end{aligned}$$

Se assumiamo $|f''| \approx |f'''|$ otteniamo che

$$\frac{\bar{h}_{1p}^2}{\bar{h}_{dc}^3} \approx \frac{1}{3} \quad \Longrightarrow \quad \bar{h}_{1p} \approx \frac{1}{3}\bar{h}_{dc}^{3/2} \tag{4.10}$$

Poichè $h \ll 1$ questa relazione ci dice che $\bar{h}_{1p} < \bar{h}_{dc}$, ossia per calcolare $f'(x)$ con la massima accuratezza usando lo schema ad un punto devo usare un incremento più piccolo di quello che userei con lo schema alle differenze centrali.

4.1.2 Derivate di ordine superiore

Con la stessa tecnica si possono ottenere approssimazioni per le derivate di ordine superiore. Ci limiteremo alla derivata seconda per la quale sommando gli sviluppi per $x+h$ e per $x-h$ e sottraendo 2 volte $f(x)$ si può ottenere

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + O(h^2) \qquad \text{differenza centrale} \tag{4.11}$$

Nello scrivere questa formula in un computer non conviene tradurla come è scritta perchè esalta l'errore di round-off. Infatti prima il computer calcola $f(x+h)+f(x-h)$ e $2f(x)$ e poi sottrae le due quantità. Conviene invece codificare la seguente espressione

$$f''(x) = \frac{[f(x+h) - f(x)] + [f(x-h) - f(x)]}{h^2} \tag{4.12}$$

Si possono ottenere formule ad ordine pi` elevato di precisione. Per esempio all'ordine 4 si ottiene

$$f''(x) = \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2} + O(h^4) \tag{4.13}$$

Considerazioni analoghe a quelle svolte per il caso della derivata prima possono essere fatte anche in questo caso per l'anallisi degli errori. In particolare sarà importante determinare l'incremento ottimale che minimizza l'errore totale per il dato schema e la data precisione della macchina.

4.2 Integrazione numerica in una dimensione

L'integrazione di una funzione nota richiede una certa abilità analitica e a volte non è la primitiva non ha una forma espicita. Numericamente invece l'integrazione è una procedura abbastanza semplice e robusta nel senso che, se la funzione non ha patologie (ossia è integrabile) e se posso generarla in un numero qualsiasi di punti nell'intervallo di integrazione allora posso senz'altro ottenere una stima più o meno accurata del suo integrale.
Nella definizione di Riemann, l'integrale di $f(x)$ in $[a,b]$ è il limite della somma delle aree per l'ampiezza degli intervallini che va a zero

$$\int_a^b f(x)dx = \lim_{h\to 0} \sum_{i=1}^{(b-a)/h} hf(x_i) \tag{4.14}$$

Numericamente l'integrale della funzione è approssimato da una somma finita che è l'equivalente della definizione di Riemann con ampiezza degli intervalli finita

$$I = \int_a^b f(x)dx \approx I_N = \sum_{i=1}^{N} w_i f(x_i) \tag{4.15}$$

dove w_i sono opportuni pesi e x_i opportuni punti che dipendono dall'algoritmo scelto. In generale si può dimostrare che analiticamente

$$\lim_{N\to\infty} I_N = I \tag{4.16}$$

Numericamente invece la precisione cresce con N finchè l'errore di round-off non domina la precisione dell'algoritmo. Poichè l'approssimazione più appropriata dipende dal comportamento della specifica funzione in esame e dall'intervallo di integrazione non esiste in generale uno

schema migliore.
Negli schemi più semplici, l'integrando è approssimato con i primi pochi termini dello sviluppo di Taylor della funzione che viene poi integrato termine a termine. A meno che l'integrando non abbia un comportamento patologico in ogni integrvallo, aumentando il numero di termini nello sviluppo si ottengono schemi più accurati. In questi metodi "Newton-Cotes", l'intervallo di integrazione è diviso in intervalli di stessa ampiezza (punti equidistanti) e l'integrando è valutato a punti equidistanti.
Questo tipo di algoritmi include

- il metodo dei trapezi (1^o ordine) $\{w_i\} = h\{1/2, 1, \ldots, 1, 1/2\}$
- il metodo di Simpson (2^o ordine) $\{w_i\} = h/3\{1, 4, 2, 4, 2, \ldots, 2, 4, 2, 4, 1\}$

Metodo dei trapezi

Consideriamo l'intervallo $[a, b]$ e N punti equidistanti $x_i \in [a, b]$ definiti da

$$\begin{aligned} x_i &= a + h(i-1) \qquad\qquad i = (1, \ldots, N) \qquad \text{con} \\ h &= \frac{b-a}{N-1} \end{aligned}$$

Lo schema consiste nell'approssimare la funzione nel generico intervallo $[x_i, x_{i+1}]$ con una retta che passa per i punti $(x_i, f_i); (x_{i+1}, f_{i+1})$ con $f_i = f(x_i)$

$$\int_{x_i}^{x_{i+1}} dx\ f(x) \approx \int_{x_i}^{x_i+h} dx\ \left(f_i + \frac{f_{i+1} - f_i}{h} x\right) = f_i\ h + \frac{(f_{i+1} - f_i)}{h} \frac{h^2}{2} = \frac{h}{2}(f_{i+1} + f_i) \quad (4.17)$$

in modo che l'integrale in $[a, b]$ risulta

$$I \approx I_N = \frac{h}{2}(f_1 + f_N) + h \sum_{i=2}^{N-1} f_i \quad \Longrightarrow \quad w_i = h\{\frac{1}{2}, 1, \ldots, 1, \frac{1}{2}\} \quad (4.18)$$

Metodo di Simpson

In questo metodo si approssima la funzione in un intervallo con una parabola $p(x) = \alpha\ x^2 + \beta\ x + \gamma$. Consideriamo un cambio di variabile che porti il centro dell'intervallo nell'origine. Si ha

$$\int_{-h}^{h} dy f(y) \approx \int_{-h}^{h} dy\ \left(\alpha\ y^2 + \beta\ y + \gamma\right) = \frac{2\alpha}{3} h^3 + 2\gamma\ h \quad (4.19)$$

Poichè ho 3 parametri da fissare devo imporre che la parabola sia uguale alla funzione da approssimare in 3 punti diversi. Ho bisogno quindi di considerare due intervalli adiacenti e imporre che

$$\begin{aligned} f(-h) &= \alpha h^2 - \beta h + \gamma \\ f(0) &= \gamma \\ f(h) &= \alpha h^2 + \beta h + \gamma \end{aligned}$$

da cui segue

$$\begin{aligned} \gamma &= f(0) \\ \alpha &= \frac{1}{h^2}\left(\frac{f(h) - 2f(0) + f(-h)}{2}\right) \end{aligned}$$

Sostituendo queste relazioni in eq. (**??**) si ottiene

$$\int_{-h}^{h} dy f(y) \approx \frac{h}{3} f(h) + \frac{4h}{3} f(0) + \frac{h}{3} f(-h) \tag{4.20}$$

Questa relazione vale per ogni coppia di intervalli adiacenti di ampiezza h. Il valore dell'integrale in $[a, b]$ sarà quindi dato da

$$I \approx I_N = \frac{h}{3} f_1 + \frac{4h}{3} f_2 + \frac{2h}{3} f_3 + \ldots + \frac{2h}{3} f_{N-2} + \frac{4h}{3} f_{N-1} + \frac{h}{3} f_N \tag{4.21}$$

ossia

$$I \approx I_N = \sum_{i=1}^{N} w_i f_i \qquad \text{con} \qquad w_i = \{\frac{h}{3}, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \ldots, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \frac{h}{3}\} \tag{4.22}$$

N deve essere dispari in modo da avere un numero pari di intervalli. Si può verificare la seguente regola di somma

$$\sum_{i=1}^{N} w_i = h(N-1) \tag{4.23}$$

4.2.1 Analisi degli errori

In generale si vuole scegliere uno schema di integrazione che fornisca una risposta accurata con il minimo numero di punti di griglia. Per ottenere un'ordine di grandezza dell'errore di troncamento consideriamo il singolo intervallo e sviluppiamo la funzione integranda in serie di Taylor rispetto al centro dell'intervallo

$$\begin{aligned} I_i = \int_{-h/2}^{h/2} dx\ f(x) &\simeq \int_{-h/2}^{h/2} dx\ \left[f(0) + x f'(0) + \frac{x^2}{2} f''(0) + \frac{x^3}{3!} f'''(0) + \frac{x^4}{4!} f^{iv}(0) + \ldots \right] \\ &= h\ f(0) + \frac{h^3}{48} f''(0) + \frac{h^5}{120 \cdot 2^5} f^{(iv)}(0) + O(h^7) \end{aligned}$$

dove i termini con le potenze dispari nell'integrando danno contiubuto nullo perchè integro su un intervallo simmetrico rispetto all'origine.
Nel metodo dei trapezi sostituiamo la funzione integranda con la retta e quindi l'integrale sul singolo intervallo è stimato con una precisione $O(h^3)$. Si può pensare di commettere lo stesso errore su tutti gli intervalli e quindi l'errore di troncamento sul risultato totale sarà

$$\text{Trapezi} \qquad E_{tr}^{tot} \sim |f''|\ O(Nh^3) = |f''|\ O\left(\frac{(b-a)^3}{N^2}\right) \tag{4.24}$$

Nel metodo di Simpson invece l'errore di troncamento sul singolo intervallo è $O(h^5)$ e quindi l'errore di troncamento sul valore dell'integrale totale è

$$\text{Simpson} \qquad E_{tr}^{tot} \sim |f^{(iv)}|\ O(Nh^5) = |f^{(iv)}|\ O\left(\frac{(b-a)^5}{N^4}\right) \tag{4.25}$$

L'errore di troncamento nel metodo di Simpson segue una legge di potenza più favorevole che nel metodo dei trapezi (assumendo che le derivate siano maggiorabili con una costante finita in $[a, b]$). Possiamo definire l'errore relativo di troncamento dividendo E_{tr}^{tot} per il valore della funzione

$$\epsilon_{tr}^{tot} = \frac{E_{tr}^{tot}}{|f|} \tag{4.26}$$

Consideriamo ora il contributo dell'error di round-off. Facciamo l'ipotesi che, dopo N calcoli, l'errore relativo casuale compiuto dalla macchina sia accumulato secondo la relazione

$$\epsilon_{ro} \sim \sqrt{N}\epsilon_m \tag{4.27}$$

dove ϵ_m è precisione della macchina sulla singola computazione, e $\sqrt{N}$ indica che l'errore si propaga secondo la legge di propagazione degli errori casuali. L'errore totale sul calcolo dell'integrale sarà quindi

$$\epsilon_{tot} = \epsilon_{tr}^{tot} + \epsilon_{ro} \tag{4.28}$$

Come in precedenza assumiamo che il minimo di questo errore si ottiene per un valore $\bar{N}$ a cui l'errore di trancamento eguaglia quello di round-off

$$\epsilon_{ro} = \epsilon_{tr}^{tot} \tag{4.29}$$

Supponiamo che i maggioranti di f e delle sue derivate di ordine superiore siano dello stesso ordine di grandezza in $[a, b]$, e riscaliamo l'intervallo in $[0, 1]$ (basta una traslazione e un cambio di scala). Così facendo otteniamo $h = 1/N$. Abbiamo quindi

- **Trapezi**

$$\epsilon_{ro} = \epsilon_{tr}^{tot} \Longrightarrow \sqrt{\bar{N}}\epsilon_m \simeq \frac{|f''|(b-a)^3}{|f|\bar{N}^2} \simeq \frac{1}{\bar{N}^2} \sim h^2 \Longrightarrow \bar{N} \sim \left(\frac{1}{\epsilon_m}\right)^{\frac{2}{5}} \tag{4.30}$$

  che ha le due consequenze

  1.
  $$\bar{N} = \begin{cases} \left(\frac{1}{10^{-7}}\right)^{\frac{2}{5}} \simeq 631 & \text{precisione singola} \\ \left(\frac{1}{10^{-15}}\right)^{\frac{2}{5}} \simeq 10^6 & \text{precisione doppia} \end{cases} \tag{4.31}$$

  2.
  $$\epsilon_{ro}(\bar{N}) \approx \sqrt{\bar{N}}\epsilon_m = \begin{cases} 3 \times 10^{-6} & \text{precisione singola} \\ 10^{-12} & \text{precisione doppia} \end{cases} \tag{4.32}$$

- **Simpson**

$$\epsilon_{ro} = \epsilon_{tr}^{tot} \Longrightarrow \sqrt{\bar{N}}\epsilon_m \simeq \frac{|f^{(iv)}|(b-a)^5}{|f|\bar{N}^4} \simeq \frac{1}{\bar{N}^4} \sim h^4 \Longrightarrow \bar{N} \sim \left(\frac{1}{\epsilon_m}\right)^{\frac{2}{9}} \tag{4.33}$$

  che ha le due consequenze

  1.
  $$\bar{N} = \begin{cases} \left(\frac{1}{10^{-7}}\right)^{\frac{2}{9}} \simeq 36 & \text{precisione singola} \\ \left(\frac{1}{10^{-15}}\right)^{\frac{2}{9}} \simeq 2154 & \text{precisione doppia} \end{cases} \tag{4.34}$$

  2.
  $$\epsilon_{ro}(\bar{N}) \approx \sqrt{\bar{N}}\epsilon_m = \begin{cases} 6 \times 10^{-7} & \text{precisione singola} \\ 5 \times 10^{-14} & \text{precisione doppia} \end{cases} \tag{4.35}$$

- **Conclusioni**
  - il metodo di Simpson è più preciso del metodo dei trapezi

- per ottenere il risultato con l'errore minimo non bisogna andare nel limite $N \to \infty$ come si potrebbe pensare considerando solo l'errore di troncamento, ma anzi bisogna mantenerlo piuttosto piccolo, specie in singola precisione.
- usando Simpson è possibile ottenere un valore dell'errore vicino alla precisione della macchina, mentre con il metodo dei trapezi si perde precisone anche usando la doppia precisione

Capitolo 5

Equazioni differenziali ordinarie

In fisica, come in mlti altri campi della scienza, le Equazioni Differenziali Ordinarie (EDO) sono di importanza fondamentale perchè descrivono, almeno nella moderna visione del calcolo infinitesimale, l'evoluzione temporale (ma non solo) dei sistemi, ossia le leggi del moto.

Un'equazione differenziale di ordine n può sempre essere ridotta ad un sistema di n equazioni differenziali ordinarie di ordine 1. Cominciamo quindi ad occuparci di queste e affronteremo in seguito la soluzione dei sistemi di EDO. Consideriamo l'equazione

$$\frac{dx}{dt} = f(t, x) \tag{5.1}$$

Trattandosi di una equazione del primo ordine, la soluzione generale sar una famiglia di soluzioni dipendenti da un parametro. La particolare soluzione, in questa famiglia, si otterrà imponendo la condizione iniziale data: $x(0) = a$.

Consideriamo ad esempio l'equazione a coefficienti costanti $\dot{x} = x + t$. Questa equazione è lineare e quindi la soluzione è una opportuna combinazione lineare di una soluzione dell'equazione omogenea associata, $\dot{x} - x = 0$ e di una soluzione particolare dell'equazione forzata. La soluzione dell'omogenea è

$$x(t) = x_0 e^t$$

Per la soluzione della forzata ipotizziamo un comportamento polinomiale:

$$x(t) = \alpha t^2 + \beta t + \gamma$$

che sostituita nell'equazione di partenza dà

$$2\alpha t + \beta - \alpha t^2 - \beta t - \gamma = t$$

da cui ricaviamo

$$\begin{aligned} \alpha &= 0 \\ \beta &= -1 \\ \gamma &= -1 \end{aligned}$$

La soluzione del problema $\dot{x} - x = t$ è quindi

$$x(t) = x_0 e^t - t - 1$$

Imponendo la condizione iniziale $x(0) = a$, da cui si ottiene $x_0 - 1 = a$ e quindi $x_0 = 1 + a$ si ottiene

$$x(t) = (1 + a)e^t - t - 1$$

Il primo metodo che può venire in mente per affrontare la soluzione numerica dell'equazione del primo ordine: $(\dot{x} = f(t,x); x(0) = a)$, è quello di sviluppare la soluzione a tempi positivi a partire dallo sviluppo in serie di potenze intorno alla condizione iniziale

$$x(h) = x(0) + hf[x, x(0)] + \frac{h^2}{2}\left(\dot{f}\right)_0 + \frac{h^3}{6}\left(\ddot{f}\right)_0 + \cdots \tag{5.2}$$

Questa procedura però non fornice un metodo di integrazione numerica del problema perchè:

- non sappiamo a priori il raggio di convergenza della serie di potenze che dipende dalla forma specifica di f(t,x) e delle sue derivate temporali (totali) e anche dal punto intorno a cui si sviluppa;
- pur supponendo che la serie converga, potrebbe essere a convergenza lenta per h abbastanza grande e sarebbe necessario considerare molti temini della serie;
- è necessario derivare f(t,x) molte volte ed è quindi molto complicato scrivere un algoritmo generale in cui il cambio della equazione comporta cambiare poche righe di codice;
- non è facile conoscere l'entità dell'errore troncando ad un certo ordine n perchè se h $O(1)$ quello che conta è il modulo della derivate di ordine $n+1$ in un punto interno dell'intervallo $[0, h]$ mentre si conosce solo la derivata (n+1)-esima in 0.

La *località* temporale delle EDO, ossia il fatto che la soluzione al tempo immediatamente vicino al presente dipende solo dal presente, ci permette però di utilizzare metodi ricorsivi per integrare le EDO. In questi metodi si avanza nel tempo a piccoli passi e si itera questa procedura. Se l'accuratezza nel singolo passo è sufficientemente elevata e lo schema è stabile, l'applicazione ripetuta dell'algoritmo può fornire la soluzione a tempi arbitrariamente grandi. Gli algoritmi che presenteremo qui sono tutti di tipo ricorsivo.

5.1 Algoritmo di Eulero

E' l'algoritmo ricorsivo più semplice e deriva dall'uso della formula ad un punto per la derivate prima di una funzione (forward-difference), ossia sullo sviluppo di Taylor troncato al primo ordine:

$$x(t+h) = x(t) + h\dot{x}(t) + O(h^2) = x(t) + hf(t, x(t)) + O(h^2) \tag{5.3}$$

Si vede che l'errore di troncamento è di ordine h^2 $(O(h^2))$. Tuttavia come vedremo in seguito, l'applicazione ricorsiva dello schema genera una accumulazione di errori. L'errore commesso al tempo T=Nh, ossia dopo N applicazioni dello schema sarà solo $O(h)$. Questa è una proprietà generale degli schemi iterativi: l'**errore globale** è sempre di un ordine di grandezza inferiore rispetto all'errore **locale**, ossia di troncamento dello schema.

Deriviamo ora un criterio di convergenza e stabilità per questo algoritmo. Supponiamo di conoscere la soluzione esatta (analitica) di un dato problema e indichiamo con ϵ_n la differenza tra la soluzione numerica all'iterazione n-esima e la soluzione esatta al tempo t=nh (x_n). Applicando ulteriormente lo schema di Eulero si ottiene:

$$x_{n+1} + \epsilon_{n+1} = x_n + \epsilon_n + hf(nh, x_n + \epsilon_n) = x_n + \epsilon_n + hf(nh, x_x) + h\epsilon_n \left(\frac{\partial f}{\partial x}\right)_{(nh, x_n)} + O(\epsilon_n^2) \tag{5.4}$$

da cui evinciamo che l'errore al passo (n+1) è proporzionale all'errore al passo n

$$\epsilon_{n+1} \simeq \left[1 + h\left(\frac{\partial f}{\partial x}\right)_{(nh, x_n)}\right] \epsilon_n + O(\epsilon_n^2) \tag{5.5}$$

Quindi l'andamento dell'errore cresce o dimunisce nel corso delle iterazioni seconda se il termine tra parentesi quadre è maggiore o minore di 1. Lo schema si dirà localmente **stabile** se

$$\left|\left[1 + h\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)}\right]\right| < 1 \tag{5.6}$$

ossia

$$-1 < 1 + h\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)} < 1 \tag{5.7}$$

Poiché $h > 0$ per definizione (cerco la soluzione a tempi positivi), la disugiaglianza a destra sarà verificata solo se $\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)} < 0$. Per soddisfare la disuguaglianza a sinistra deve invece valere

$$h < 2\left[\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)}\right]^{-1} \tag{5.8}$$

che impone un limite di stabilità su h.
Vediamo come esempio, tre casi importanti, semplici ed interessanti

$$\begin{aligned}
&\dot{x} + \alpha x = 0 \Longrightarrow x(t) = x(0)e^{-\alpha t} && \text{decrescita esponenziale}\\
&\dot{x} - \alpha x = 0 \Longrightarrow x(t) = x(0)e^{+\alpha t} && \text{crescita esponenziale}\\
&\dot{x} \pm \imath\alpha x = 0 \Longrightarrow x(t) = x(0)e^{\pm\imath\alpha t} && \text{oscillazione}
\end{aligned}$$

nel caso di oscillazioni $\left(\frac{\partial f}{\partial x}\right)$ è complessa e quindi conviene considerare il modulo quadrato di $\left[1 + h\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)}\right]$. La condizione di stabilità in questo caso è

$$\left|1 + h\left(\frac{\partial f}{\partial x}\right)_{(nh,x_n)}\right|^2 < 1 \tag{5.9}$$

Sostituendo la derivata esplicita nel caso oscillatorio si ottiene:

$$|1 - \imath\alpha h]^2 = 1 + (\alpha h)^2 > 1 \tag{5.10}$$

Quindi lo schema di Eulero per i moti oscillatori è sempre instabile.

Vediamo infine come l'applicazione ripetuta dell'algoritmo che ha un *errore locale* di ordine h^2 genera una accumulazione di errori e un *errore globale* di ordine h. Oltre ai criteri imposti dalla stabilità, assumiamo che il resto nell'equazione (**??**) sia limitato (ossia $\exists M > 0 : \left|\frac{\partial^2 f}{\partial x^2}\right| < M$). Con questa ipotesi possiamo scrivere

$$|\epsilon_{n+1}| \leq |1 + hK|\,|\epsilon_n| + \frac{h^2 M}{2} \tag{5.11}$$

dove K è una maggiorazione per la derivata prima di f rispetto a x: $|\partial f/\partial x| < K, K > 0$. Questa formula ricorsiva definisce uno schema alle differenze finite del secondo ordine

$$\begin{aligned}
z_{n+1} &= z_n(1 + hK) + \frac{h^2 M}{2}\\
z_0 &= 0 \qquad\qquad \text{(a t=0 l'errore è nullo)}
\end{aligned}$$

che, come è facile verificare per sostituzione, ha la soluzione

$$z_n = \frac{hM}{2K}(1 + hK)^n - \frac{hM}{2K}$$

per $hK \ll 1$ posso sostituire $(1+hK)$ con e^{hK} (in ogni caso $(1+hK) < e^{hK}(K>0)$) e ottenere

$$z_n \leq \frac{hK}{2K} e^{nhK} - \frac{hM}{2K} = \frac{hM}{2K}(e^{nhK} - 1) = \frac{hM}{2K}(e^{(t_n - t_0)K} - 1) = O(h)$$

Abbiamo quindi mostrato che lo schema di Eulero con errore locale di $O(h^2)$ ha un errore globale $O(h)$. Questa è una proprietà generale degli schemi iterativi.

5.2 Algoritmi di Runge-Kutta

Prendono il nome dai loro inventori, due matematici tedeschi. Sono algoritmi molto usati perché con un piccolo costo aggiuntivo ad iterazione raggiungono accuratezza elevata. Il metodo del 4^o ordine è tra i più usati per le EDO. Come per l'algoritmo di Eulero sono basati sullo sviluppo in serie di potenze della soluzione intorno alla condizione iniziale (della singola iterazione). Tuttavia la loro derivazione è algebricamente complicata. Deriviamo qui solo il metodo al 2^o ordine per illustrare lo spirito della derivazione.

Consideriamo l'EDO:

$$\dot{x} = f(t, x)$$

scriviamo l'incremento di x come una media pesata di due stime, k_1, k_2, di Δx:

$$\begin{cases} x_{n+1} = x_n + ak_1 + bk2 \\ k_1 = hf(t_n, x_n) \\ k_2 = hf(t_n + \alpha h, x_n + \beta k_1) \end{cases} \tag{5.12}$$

con $t_n = nh, x_n = x(t_n)$. Possiamo pensare a k_1 e k_2 come stime del cambiamento in x all'avanzare di h del tempo t perchè sono il prodotto del passo temporale h per una stime della derivata della curva soluzione dell'equazione. Il metodo Runge-Kutta usa sempre come prima stima di Δx la stima di Eulero k_1. Il problema è scegliere i quattro parametri (a, b, α, β) in maniera che lo schema riproduca nel miglior modo possibile lo sviluppo di Taylor della soluzione in cui le derivate temporali di $x(t)$ sono ottenute tramite quelle di $f(t, x)$. Abbiamo

$$x_{n+1} = x_n + hf(t_n, x_n) + \frac{h^2}{2}\left(\frac{df}{dt}\right)_n + O(h^3) \tag{5.13}$$

Sostituiamo l'espressione per la derivata totale di f rispetto al tempo

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\frac{dx}{dt} = f_t + f_x f \tag{5.14}$$

ottenendo

$$x_{n+1} = x_n + hf(t_n, x_n) + \frac{h^2}{2}\left(f_t + f_x f\right)_n + O(h^3) \tag{5.15}$$

Con lo schema RK2 l'equazione di evoluzione è

$$x_{n+1} = x_n + ahf_n + bhf(t_n + \alpha h, x_n + \beta h f_n) \tag{5.16}$$

dove $f_n = f(t_n, x_n)$. Per confrontare queste due equazioni sviluppiamo f(t,x) in eq.(**??**) in serie di Taylor in t e x, intorno a (t_n, x_n) e fermiamoci al 1^o ordine

$$f(t_n + \alpha h, x_n + \beta h f_n) \simeq f_n + \alpha h \left(\partial_t f\right)_n + \beta h f_n \left(\partial_x f\right)_n \tag{5.17}$$

Sostituendo questa espressione nell'eq.(**??**) si ottiene

$$x_{n+1} = x_n + ahf_n + hb\left[f_n + \alpha h f_t + \beta h f f_x\right]_n = x_n + hf_n(a+b) + h^2\left[\alpha b f_t + \beta b f f_x\right]_n \tag{5.18}$$

che confrontata all'eq.(refeq:taylor) fornisce le seguenti tre equazioni

$$\begin{cases} a+b=1 \\ \alpha b = \frac{1}{2} \\ \beta b = \frac{1}{2} \end{cases} \tag{5.19}$$

Abbiamo quindi 4 incognite e solo 3 equazioni quindi abbiamo una possibile famiglia di algoritmi che soddisfanno lo siluppo di Taylor all'ordine 2. La scelta standard è $a = b = 1/2; \alpha = \beta = 1$ che fornisce

$$x_{n+1} = x_n + \frac{1}{2}hf_n + \frac{1}{2}hf(t_n + h, x_n + hf_n) \tag{5.20}$$

Questa equazione è uno schema di Eulero modificato in cui la forza è la media aritmetica della derivata nel punto iniziale e di quella nel punto finale (predictor-corrector).

La derivazione dello schema al 4^o ordine è simile in spirito ma algebricamente complicata. Si ottengono 11 equazioni in 13 incognite. Scegliendo il liberamente il valore di due parametri si ottiene una soluzione unica per gli altri parametri. La scelta pù comune fornisce l'algoritmo

$$\begin{cases} x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k2 + 2k_3 + k_4) + O(h^5) \\ k_1 = hf_n \\ k_2 = hf(t_n + \frac{h}{2}, x_n + \frac{1}{2}k_1) \\ k_3 = hf(t_n + \frac{h}{2}, x_n + \frac{1}{2}k_2) \\ k_4 = hf(t_n + h, x_n + k_3) \end{cases} \tag{5.21}$$

L'errore locale di questo schema è $O(h^5)$ quello globale $O(h^4)$. Sono necessari quattro calcoli di f ad ogni iterazione ma in genere l'incremento che si può scegliere è molto maggiore che negli altri schemi e quindi l'efficienza globale dello scheme ne risulta molto aumentata.

5.3 EDO del 2^o ordine: algoritmo di Eulero-Cromer

Consideriamo il problema

$$\begin{cases} \ddot{x} = f(t, x, \dot{x}) \\ x(0) = a \\ \dot{x}(0) = b \end{cases} \tag{5.22}$$

Come già detto possiamo sempre riscrivere una equazione del secondo ordine come un sistema di due equazioni del primo ordine

$$\begin{cases} \dot{x} = v(t) \\ \dot{v} = f(t, x, v) \\ x(0) = a \\ v(0) = b \end{cases} \tag{5.23}$$

Ciascuna di queste equazioni può essere integrata con l'algoritmo di Eulero ottenendo il seguente schema

$$\begin{cases} x_{n+1} = x_n + hv_n \\ v_{n+1} = v_n + hf(t_n, x_n, v_n) \\ x(0) = a \\ v(0) = b \end{cases} \tag{5.24}$$

Uno schema in generale più stabile di Eulero è il cosidetto Eulero-Cromer in cui si inverte l'ordine delle equazioni e si sostituisce a v_n nella seconda equazione il valore di v_{n+1} ottenuta propagando la prima equazione (simile al metodo di Gauss-Seidel per le matrici)

$$\begin{cases} v_{n+1} = v_n + hf(t_n, x_n, v_n) \\ x_{n+1} = x_n + hv_{n+1} \\ x(0) = a \\ v(0) = b \end{cases} \tag{5.25}$$

Questo metodo è stabile anche nel caso di sistemi oscillanti (per i quali Eulero non è stabile) ma rimane di $O(h)$.

5.4 Algoritmi di Verlet e Velocity-Verlet

Consideriamo problemi descrivibili con EDO del 2^o ordine in cui la forzante non dipende dalla velocità

$$\ddot{x} = f(x)$$

Problemi di questo tipo si incontrano in meccanica. Se usiamo lo schema alle differenze centrali per rappresentare la derivata seconda

$$\ddot{x} = \frac{x(t+h) - 2x(t) + x(t-h)}{h^2} + O(h^2)$$

possiamo riscrivere l'equazione differeziale risolvendo per $x(t+h) = x_{n+1}$

$$x_{n+1} = 2x_n - x_{n-1} + h^2 f(x_n) + O(h^4) \tag{5.26}$$

quindi è possibile avanzare la soluzione di un passo conoscendo la soluzione al passo attuale e al passo precedente, e la forza al passo attuale.
La velocità $v_n = v(t) = \dot{x}(t)$ può essere ottenuta con lo schema alle differenze centrali di ordine $O(h^2)$

$$v_n = \dot{x}_n = \frac{x_{n+1} - x_{n-1}}{2h} + O(h^2) \tag{5.27}$$

Notare che:

- la precisione sulla velocità è due ordini di grandezza più piccola che sulla posizione. Tuttavia, poiché la forza non dipende dalla velocità, questo errore non si propaga sulla soluzione per la posizione. Diciamo quindi che l'algoritmo di Verlet è di ordine 4.
- la velocità è anche affetta dall'errore di round-off se $h \to 0$. Di nuovo questo errore non si propaga sulla soluzione per la posizione
- questo algoritmo non è self-starting, ossia non basta conoscere la condizione iniziale usuale (x(0), v(0)) per poter cominciare l'integrazione. Bisogna invece ottenere la posizione al tempo $-h$ per poter utilizzate le eq.(**??**)-(**??**). Si può ad esempio generare questa posizione con Eulero: $x_{-1} = x_0 - hv_0 + h^2 f(x_0)/2$

Per rimediare a questi inconvenienti si può riscrivere lo stesso algoritmo in una forma diversa ma algebricamente identica, note come Velocity Verlet

$$\begin{cases} x_{n+1} = x_n + hv_{n+1} + \frac{h^2}{2} f(x_n) + O(h^3) \\ v_{n+1} = v_n + \frac{h}{2} \left[f(x_n) + f(x_{n+1}) \right] + O(h^3) \\ x(0) = a \\ v(0) = b \end{cases} \tag{5.28}$$

In questa forma l'algoritmo è self-starting e la soluzione non è affetta dall'errore di round-off. Deriviamo ora questa forma a partire dalle eq. (**??**)-(**??**).
Deriviamo ora le equazioni (**??**) dalle equazioni (**??**)-(**??**). Consideriamo dapprima l'equazione (**??**) e sommiamo e sottraiamo $x_{n+1}/2$ nel membro di destra

$$x_{n+1} = 2x_n + \frac{1}{2}\left(x_{n+1} - x_{n-1}\right) - \frac{1}{2}x_{n+1} - \frac{1}{2}x_{n-1} + h^2 f(x_n) \tag{5.29}$$

dalla equazione (**??**) il termine tra parantesi a destra è uguale a hv_n. Inoltre dall'equazione (**??**) otteniamo $h^2 f(x_n) = (x_{n+1} - 2x_n + x_{n-1})/2$ e, sostituendo nell'equazione precedente otteniamo la prima delle equazioni (**??**). Per ottenere la seconda scriviamo (dall'eq.(**??**))

$$v_{n+1} = \frac{x_{n+2} - x_n}{2h} = \frac{2x_{n+1} - 2x_n + h^2 f(x_{n+1})}{2h} = \frac{1}{h}\left(x_{n+1} - x_n + \frac{h^2}{2} f(x_{n+1})\right) \tag{5.30}$$

Sostituendo $x_n = (x_{n+1} + xn - 1)/2 - h^2 f_n/2$ (da (**??**)), si ottiene

$$v_{n+1} = \frac{1}{h}\left(x_{n+1} - \frac{x_{n+1} + x_{n-1}}{2}\right) + \frac{h}{2}[f_n + f_{n+1}] = v_n + \frac{h}{2}[f_n + f_{n+1}] \tag{5.31}$$

che è appunto l'equazione che cercavamo.

Di tutti gli schemi che studieremo, l'algoritmo di Verlet è particolarmente appropriato ai sistemi meccanici isolati (Hamiltoniani) perchè è reversibile sotto inversione del segno del tempo (simmetrico in t - time reversal invariant) e in più è **simplettico**, ossia conserva il volume dello spazio delle fasi del sistema, come la vera dinamica hamiltoniana.

5.5 Algoritmo di Runge-Kutta per EDO del 2^o ordine

E' l'estensione diretta dell'algoritmo descritto per le EDO del 1^o ordine. Scriviamo quindi solo l'espressione esplicita senza ripeterne la derivazione. Inoltre consideriamo il caso più generale di sistemi a dimensione fisica maggiore di 1, nei quali la posizione e velocità sono vettori. Consideriamo il sistema di due equazioni differenziali vettoriali accoppiate della forma

$$\begin{aligned} \dot{\vec{x}} &= \vec{g}(t, \vec{x}, \vec{v}) \\ \dot{\vec{v}} &= \vec{f}(t, \vec{x}, \vec{v}) \end{aligned} \tag{5.32}$$

La forma esplicita dell'algoritmo RK4 per tale sistema è

$$\begin{aligned} \vec{x}_{n+1} &= \vec{x}_n + \frac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \\ \vec{v}_{n+1} &= \vec{v}_n + \frac{1}{6}(\vec{j}_1 + 2\vec{j}_2 + 2\vec{j}_3 + \vec{j}_4) \end{aligned} \tag{5.33}$$

dove

$$\left\{ \begin{array}{l} \vec{k}_1 = h\vec{g}(t_n, \vec{x}_n, \vec{v}_n) \\ \vec{j}_1 = h\vec{f}(t_n, \vec{x}_n, \vec{v}_n) \\ \\ \vec{k}_2 = h\vec{g}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_1}{2}, \vec{v}_n + \frac{\vec{j}_1}{2}) \\ \vec{j}_2 = h\vec{f}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_1}{2}, \vec{v}_n + \frac{\vec{j}_1}{2}) \\ \\ \vec{k}_3 = h\vec{g}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_2}{2}, \vec{v}_n + \frac{\vec{j}_2}{2}) \\ \vec{j}_3 = h\vec{f}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_2}{2}, \vec{v}_n + \frac{\vec{j}_2}{2}) \\ \\ \vec{k}_4 = h\vec{g}(t_n + h, \vec{x}_n + \vec{k}_3, \vec{v}_n + \vec{j}_3) \\ \vec{j}_4 = h\vec{f}(t_n + h, \vec{x}_n + \vec{k}_3, \vec{v}_n + \vec{j}_3) \end{array} \right. \tag{5.34}$$

Nel caso di dinamica hamiltoniana in cui $\vec{g}(t, \vec{x}, \vec{v}) = \vec{v}$ i vari termini assumono la forma esplicita

$$\begin{cases} \vec{k}_1 = h\vec{v}_n \\ \vec{j}_1 = h\vec{f}(t_n, \vec{x}_n, \vec{v}_n) \\ \\ \vec{k}_2 = h\left(\vec{v}_n + \frac{\vec{j}_1}{2}\right) \\ \vec{j}_2 = h\vec{f}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_1}{2}, \vec{v}_n + \frac{\vec{j}_1}{2}) \\ \\ \vec{k}_3 = h\left(\vec{v}_n + \frac{\vec{j}_2}{2}\right) \\ \vec{j}_3 = h\vec{f}(t_n + \frac{h}{2}, \vec{x}_n + \frac{\vec{k}_2}{2}, \vec{v}_n + \frac{\vec{j}_2}{2}) \\ \\ \vec{k}_4 = h\left(\vec{v}_n + \vec{j}_3\right) \\ \vec{j}_4 = h\vec{f}(t_n + h, \vec{x}_n + \vec{k}_3, \vec{v}_n + \vec{j}_3) \end{cases} \tag{5.35}$$

.