# Fisica Computazionale

Adolfo Avella

25 Febbraio 2018

# Indice

# Parte I

# Teoria

# Capitolo 1

# Fisica Computazionale

Cos'è la Fisica Computazionale? Come si relaziona con la Fisica Teorica e Sperimentale e con il Metodo Sperimentale?

## 1.1 Definizione di Fisica Computazionale [1, Cap. 1, Sez. 1.1]

La Fisica Computazionale è una materia multidisciplinare che combina aspetti di fisica, matematica applicata ed informatica e che ha come scopo la risoluzione di problemi realistici nel campo della fisica [1]. Un fisico computazionale usa gli strumenti di tipo hardware (computer: processori, memorie, ...) e software (sistemi operativi, compilatori (linguaggi di programmazione), applicazioni di gestione di testi e di dati (grafici), ...) che gli informatici sviluppano. Così come usa gli algoritmi numerici che i matematici sviluppano. Questi strumenti vengono utilizzati per risolvere problemi di fisica e la loro conoscenza approfondita (degli strumenti) è necessaria perché la risoluzione dei problemi sia scientificamente corretta.

Utilizzando il paradigma del problem-solving, la Fisica Computazionale è costituita dai seguenti elementi: il **problema** da risolvere è la descrizione di un fenomeno fisico, la **teoria** da usare/validare per descrivere il fenomeno è il modello derivante dal Metodo Sperimentale nella forma di un insieme di leggi fisiche, il **metodo** da usare per risolvere le equazioni matematiche che costituiscono le leggi fisiche è l'algoritmo numerico, l'**implementazione** concreta dell'algoritmo numerico è il codice scritto in uno dei linguaggi di programmazione noti, e la **valutazione/analisi dei risultati** ottenuti si effettua utilizzando programmi di visualizzazione (grafica) ed esplorazione dei dati numerici ottenuti.

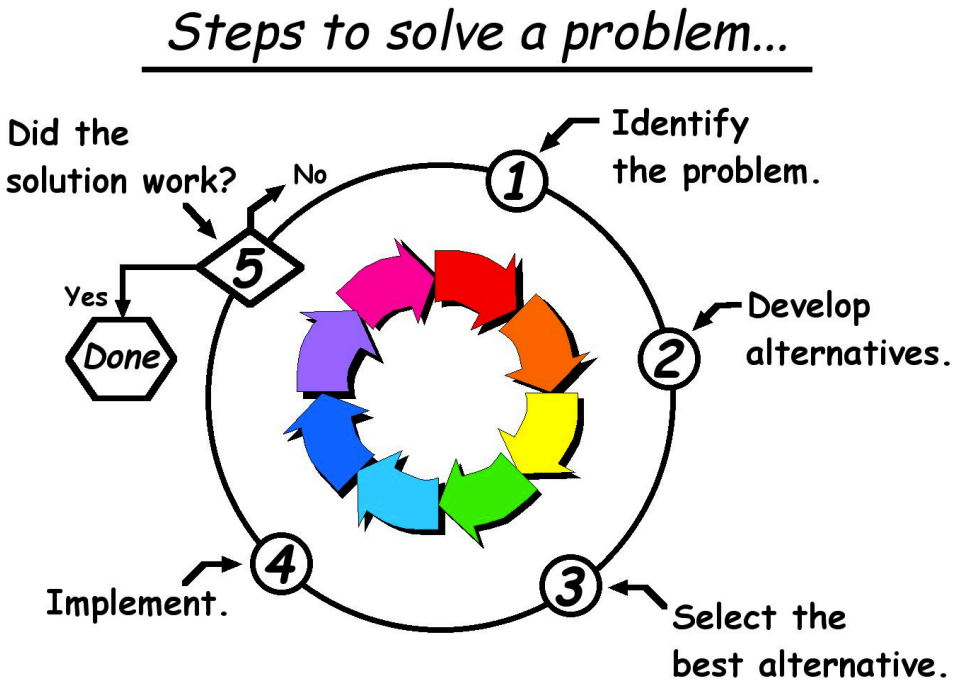


Figura 1.1: Problem Solving

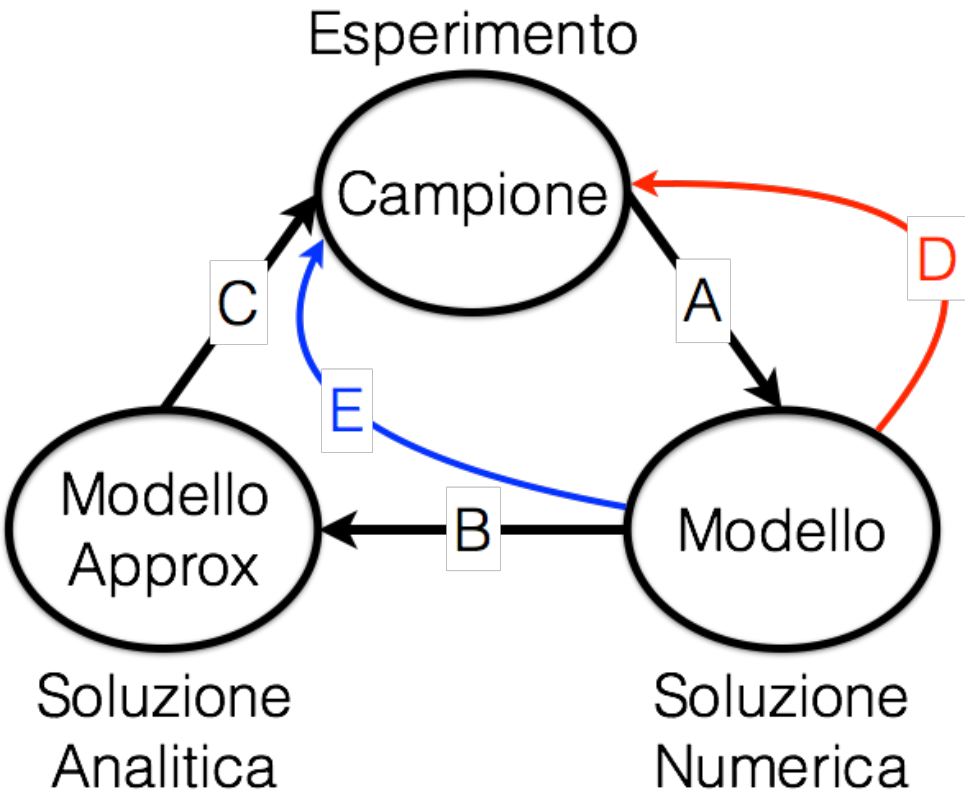


Figura 1.2: Relazione tra Campione (Esperimento), Modello (Soluzione numerica) e Modello Approssimato (Soluzione analitica)

## 1.2 Relazione con Fisica Sperimentale e Teorica [1, Cap. 1, Sez. 1.1]

La Fisica Computazionale permette di risolvere numericamente equazioni della Fisica Teorica che è impossibile risolvere analiticamente (E), se non in maniera più o meno approssimata (B), e di effettuare esperimenti virtuali (simulazioni) in condizioni fisiche estreme in cui la Fisica Sperimentale non è in grado di operare (D).

Inoltre, la Fisica Computazionale permette di validare, nei regimi dove i suoi algoritmi sono efficienti, le approssimazioni analitiche utilizzate per risolvere le equazioni di una teoria ottenuta tramite il Metodo Sperimentale nella fase di deduzione. Questo permette di scindere, nella fase di verifica del Metodo Sperimentale, la verifica del metodo analitico usato per ottenere le predizioni di una teoria (B→C) dalla verifica delle predizioni della stessa teoria (A→C). In pratica, la Fisica Computazionale permette di comprendere se una possibile difformità tra le predizioni di una teoria e l'esperimento (C) siano dovute ad un errore nel formulare la teoria (A) o ad un errore di approssimazione nel risolverla analiticamente (B) in quanto fornisce la soluzione esatta della teoria (esperimento virtuale sul modello/teoria) con cui è possibile confrontare l'approssimazione analitica e validarla o meno. Inoltre, la Fisica Computazionale permette di comprendere se un riscontro positivo tra teoria ed esperimento non sia dovuto ad un artefatto dell'approssimazione analitica, in quanto questa non restituisce una soluzione approssimata del modello compatibile con quella esatta fornita dal metodo numerico stesso. Se la Fisica Computazionale valida un'approssimazione analitica, questa può essere fiduciosamente utilizzata anche al di fuori dei regimi di validazione ed i suoi risultati possono essere direttamente confrontati con i dati sperimentali, in qualità di predizioni, con lo scopo di verificare la teoria stessa.

La scelta dell'algoritmo numerico da utilizzare, la sua implementazione sotto forma di codice, una volta scelto il linguaggio di programmazione più adatto, e l'interpretazione/elaborazione dei risultati numerici ottenuti ha molto della Fisica Teorica. Così come la simulazione numerica (esperimento virtuale), l'uso e la manutenzione di un codice (apparato di misura), la sua fase di test (calibrazione), la sua fase di esecuzione (misura) e l'analisi/controllo degli errori numerici e dei loro effetti sulla stabilità del codice e sulla precisione dei risultati numerici ottenuti (errori di misura e loro propagazione) ha molto della Fisica Sperimentale.

Tutte queste considerazioni, fanno entrare la Fisica Computazionale a pieno diritto in una moderna formulazione del Metodo Sperimentale.

# Capitolo 2

# Rappresentazione dei numeri ed errori

Dove e come conserva il computer i numeri di cui abbiamo bisogno per i nostri calcoli? Che tipo di errori possiamo incontrare nel calcolo e come possiamo tenerli sotto controllo?

## 2.1 Memoria del computer [1, Cap. 2, Sez. 2.5-2.6]

I computer sono molto potenti, ma sono finiti. Una seria difficoltà nel progettare un computer è come rappresentare un numero arbitrario usando un ammontare finito di spazio in memoria e come rimediare alle limitazioni di questa rappresentazione. Una conseguenza del fatto che le memorie fisiche del computer sono basate su sistemi fisici a semiconduttore (diodi, transistor, ...) o magnetici (ferromagneti, ...), che sono intrinsecamente bistabili (sono stabili/esistono solo in due possibili configurazioni/stati che è possibile determinare: leggere/scrivere), è che l'unità elementare di memoria del computer è il bit (b) che può assumere solo due valori 0 e 1. Questo immediatamente comporta che tutti i numeri (ed in generale le informazioni) sono conservate nella memoria del computer in formato binario, che significa sotto forma di una stringa di zeri e di uni. La connessione tra l'ordinaria forma decimale cui siamo abituati e la forma binaria cui siamo obbligati comporta il problema di stabilire una rappresentazione e di avere a che fare con le sue limitazioni.

Di solito di un computer conosciamo la lunghezza della word (parola) che indica il numero di bit utilizzati per conservare un numero in memoria. Spesso il numero di bit viene espresso in numero di byte (B), che sono agglomerati di 8 bit, e nei suoi multipli kilo(k)-, mega(M)-, giga(G)- e tera(T)- byte con la convenzione, fonte di molta confusione, che invece di avere un fattore 1,000 tra l'uno e l'altro, come avviene nella rappresentazione decimale, hanno invece un fattore $1{,}024 = 2^{10}$.

Es. 1 kB = 1,024 B = 8,192 b; 1 MB = 1,024 kB = 1,048,576 B = 8,388,608 b; 1GB = 1,024 MB = 1,048,576 kB.

## 2.2 Rappresentazione dei numeri interi [1, Cap. 2, Sez. 2.6]

Se usiamo N bit per rappresentare un numero intero, preservando il primo bit per il suo segno (0 per i positivi e 1 per i negativi), avremo la possibilità di rappresentare numeri compresi tra $-2^{N-1}+1$ e $2^{N-1}-1$ (i.e. $1\cdot2^{N-2}+1\cdot2^{N-3}+\cdots+1\cdot2^0 = \sum_{n=0}^{N-2}2^n = 2^{N-1}-1 = 1\cdot2^{N-1}+0\cdot2^{N-2}+\cdots+0\cdot2^0 - 1$)[1]. Questo porta ad avere due valori di zero (uno con segno positivo ed uno con segno negativo) che non è corretto per i numeri interi oltre che *dispendioso*: perdiamo la possibilità di rappresentare un ulteriore numero intero.

Invece, la rappresentazione usata è la seguente:

$$i = \begin{bmatrix} i_{N-1} & i_{N-2} & i_{N-3} & \cdots & i_1 & i_0 \end{bmatrix} = -i_{N-1}\cdot 2^{N-1} + \sum_{n=N-2}^{0} i_n \cdot 2^n \tag{2.1}$$

Di conseguenza, avremo i range riportati in Tabella 2.1. Es. $+7_{1B}$ = [00000111], $-7_{1B}$ = [11111001], $+127_{1B}$ = [01111111], $-127_{1B}$ = [10000001], $0_{1B}$ = [00000000], $-128_{1B}$ = [10000000], $-1_{1B}$ = [11111111]. Il comando Fortran Huge(i) restituisce il numero intero più grande, in valore assoluto, che è possibile rappresentare nella classe di rappresentazione del numero intero i.

[1] $S_{N+1} = \sum_{n=0}^{N+1} a^n = a^{N+1} + a^N + \cdots + a^2 + a + 1 = \begin{cases} a^{N+1} + S_N \\ 1 + aS_N \end{cases} \Rightarrow a^{N+1} + S_N = 1 + aS_N \Rightarrow S_N = \frac{a^{N+1}-1}{a-1}$

| word | range |
|---|---|
| 1B | -128 ÷ +127 |
| 2B | -32,768 ÷ +32,767 |
| 4B | -2,147,483,648 ÷ +2,147,483,647 |
| 8B | -9,223,372,036,854,775,808 ÷ +9,223,372,036,854,775,807 |

Tabella 2.1: Range delle rappresentazioni dei numeri interi

Questa rappresentazione permette di ottenere i numeri negativi dai positivi tramite la complementazione a 2: invertire gli zeri con gli uni e viceversa ed aggiungere 1. Es. $+7_{1B}$ = [00000111] → $-7_{1B}$ = [11111000] + [00000001] = [11111001].

Un altro modo per determinare la corretta scrittura in binario di un numero negativo è quello di comporlo in binario con un 1 nel bit più a sinistra (questo ci assicura che il numero è negativo) ed utilizzare i rimanenti N-1 bit per esprimere in binario il numero ottenuto dalla somma del numero negativo di partenza e $2^{N-1}$. Es. $-7_{1B}$ → -7 + $2^7$ = -7 + 128 = $121_{1B}$ = [01111001] → $-7_{1B}$ = [11111001].

La rilevanza fondamentale di questa rappresentazione è che permette di sommare i numeri in binario secondo le usuali regole di somma binaria (0 + 0 = 0, 1 + 0 = 1, 0 + 1 = 1, 1 + 1 = 10) senza tener conto del segno proprio della sola rappresentazione decimale e con l'unica prescrizione di non tener conto del bit N+1 a sinistra nel caso dovesse diventare 1. Questo modo di effettuare le somme porta però delle conseguenze inattese di cui è bene essere al corrente!!! Es. $-128_{1B}$ + ($+1_{1B}$) = [10000000] + [00000001] = [10000001] = $-127_{1B}$, $-1_{1B}$+ ($+1_{1B}$) = [11111111] + [00000001] = 1[00000000] = [00000000] = $0_{1B}$. !!! $+127_{1B}$ + ($+1_{1B}$) = [01111111] + [00000001] = [10000000] = $-128_{1B}$, $-128_{1B}$ + ($-1_{1B}$) = [10000000] + [11111111] = 1[01111111] = [01111111] = $+127_{1B}$ !!!

## 2.3 Rappresentazione dei numeri reali [1, Cap. 2, Sez. 2.7-2.8]

In principio, utilizzando N bit, possiamo usare la seguente rappresentazione in virgola fissa dei numeri reali:

$$r = \begin{bmatrix} r_{N-1} & r_{N-2} & r_{N-3} & \cdots & r_1 & r_0 \end{bmatrix} = (-1)^{r_{N-1}} \sum_{n=N-2}^{0} r_n \cdot 2^{n-m} \tag{2.2}$$

dove il bit in posizione N-1 dà il segno, i bit dalla posizione N-2 alla m danno le potenze positive o nulle di 2 ed i bit dalla posizione m-1 alla 0 danno le potenze negative di 2. Questa rappresentazione ha come range, in valore assoluto, $2^{-m} \leq |r| \leq 2^{-m}$ ($2^{N-1}$ - 1) = $2^{N-1-m}$ - $2^{-m}$. Questa rappresentazione introduce un errore assoluto per tutti i numeri rappresentati che è proprio dell'ordine di $2^{-m}$, quindi l'errore relativo cresce da $\frac{1}{2^{N-1}-1}$ a 1 muovendosi dal numero più grande, in valore assoluto, verso il numero più piccolo, in valore assoluto, del range. Questo è ovviamente inaccettabile, perché è ragionevole richiedere che la rappresentazione abbia un valore dell'errore relativo praticamente costante o quasi per tutti i numeri presenti nel suo range.

La soluzione è una rappresentazione in virgola mobile (floating-point) così come stabilita dallo standard IEEE 754 adottato da IEEE ed ANSI:

$$r = \begin{bmatrix} r_{N-1} & r_{N-2} & r_{N-3} & \cdots & r_1 & r_0 \end{bmatrix} = (-1)^{r_{N-1}} \cdot 2^{e-\text{bias}} \cdot (a+f) \tag{2.3}$$

dove

$$e = \sum_{n=N-2}^{m} r_n \cdot 2^{n-m} \qquad \text{bias} = 2^{\text{N}-2-\text{m}} - 1 \qquad f = \sum_{n=m-1}^{0} r_n \cdot 2^{n-m} \qquad a = 0, 1 \tag{2.4}$$

Il bit in posizione N-1 dà il segno; $e$ è l'esponente biased ed ha come range [0, $2^{N-1-m}$ - 1] così che l'esponente, a causa del bias, ha un range [$-2^{N-2-m}$ + 1, $2^{N-2-m}$]; $f$ è la mantissa ed ha come range [0, $2^{-m}$ ($2^m$ - 1) = 1 - $2^{-m}$]; a assume valore 0 o 1 a seconda di utilizzare una rappresentazione sub-normale (e = 0, f ≠ 0) o normale (0 < e < $2^{N-1-m}$ - 1), rispettivamente. Il Fortran non ammette la rappresentazione sub-normale, perché questa reintroduce il problema di un errore assoluto constante e non di uno relativo quasi-costante come desiderato.

Nel caso di una rappresentazione a 4B (8B), m è fissato dallo standard a 23 (52) così da dare i range riportati in Tabella 2.2. La rappresentazione sub-normale in grassetto non è ammessa dal Fortran.

I comandi Fortran Huge(r) e Tiny(r) restituiscono il numero reale più grande e più piccolo, rispettivamente, in valore assoluto, che è possibile rappresentare nella classe di rappresentazione del numero reale r. Stabilita una rappresentazione, se induciamo il Fortran a determinare/usare un numero superiore al corrispondente valore di Huge, il Fortran assegnerà alla variabile corrispondente il valore Infinity con l'appropriato segno (il sistema potrebbe segnalare l'occorrenza con un messaggio di warning o di errore di tipo overflow); se induciamo il Fortran

a determinare/usare un numero inferiore al corrispondente valore di Tiny, il Fortran assegnerà alla variabile corrispondente il valore 0 con l'appropriato segno (il sistema potrebbe segnalare l'occorrenza con un messaggio di warning o di errore di tipo underflow); se induciamo il Fortran ad eseguire un'operazione mal-definita (e.g. 0/0), il Fortran assegnerà alla variabile corrispondente il valore NaN (Not a Number) di tipo signaling (cioè che segnalerà l'occorrenza con un messaggio di warning o di errore) e le seguenti operazioni che coinvolgono il primo NaN restituiranno un NaN di tipo quiet (cioè che non segnalerà questa occorrenza).

## 2.4 Precisione della rappresentazione dei numeri reali [1, Cap. 2, Sez. 2.7-2.10, 2.11-2.15]

Risulta evidente che una tale rappresentazione per i numeri reali non solo porta ad un range finito dei numeri rappresentabili (così come accade per i numeri interi), ma anche ad una ben definita precisione (data dal numero di cifre significative in notazione decimale) con cui questi possono essere rappresentati. In particolare, una rappresentazione a 4B (8B) comporta un precisione di circa 6 (15) cifre decimali significative. Di fatti, la mantissa (che contiene l'informazione riguardo alle cifre significative) è nota per multipli di $2^{-m}$ e quindi per multipli di circa $1.\cdot 10^{-7}$ ($2.\cdot 10^{-16}$) e quindi con una imprecisione sulla $7^a$ ($16^a$) cifra. Il comando Fortran Precision(r) restituisce il numero di cifre decimali significative relativo alla classe di rappresentazione del numero reale r.

Questa informazione può essere ulteriormente quantificata definendo la precisione macchina $\varepsilon_m$ come il più grande numero positivo che aggiunto ad 1 ridà 1 nella rappresentazione scelta: $1._{nB} + \varepsilon_m = 1._{nB}$. Abbiamo che $\varepsilon_m$ è circa $2^{-m}$ è quindi $1.1920929\cdot 10^{-7}$ ($2.220446049250313\cdot 10^{-16}$) per una rappresentazione a 4B (8B). Il comando Fortran Epsilon(r) restituisce l'$\varepsilon_m$ relativo alla classe di rappresentazione del numero reale r.

Tentiamo di capire come sia possibile che $1._{nB} + \varepsilon_m = 1._{nB}$. Proviamo a sommare 7 e $10^{-7}$ in una rappresentazione a 4B:

$$\begin{aligned} +7._{4B} &= +\left(1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0\right) = +2^2\left(1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2}\right) = \underline{+}2^{\overline{129}-127}\left(1+\underline{1}\cdot 2^{-1}+\underline{1}\cdot 2^{-2}\right) \\ &= 0|10000001|11000000000000000000000 \end{aligned} \tag{2.5}$$

$$\begin{aligned} +10^{-7}{}_{4B} &\cong +2^{-47}\cdot 14,073,749 \\ &= +2^{-47}\,(11010110101111110010101) \\ &= +2^{-24}\,(1.10101101011111110010101) \\ &= \underline{+}2^{\overline{103}-127}\,(1+0.\underline{10101101011111110010101}) \\ &= 0|01100111|10101101011111110010101 \end{aligned} \tag{2.6}$$

Dovendo sommare le mantisse abbiamo la necessità di avere gli stessi esponenti. Possiamo far crescere di un'unità l'esponente di $10^{-7}$ shiftando a destra di un posto la sua mantissa ($2^a(2^b)$=$2^{a+1}(2^{b-1})$) (!!! manteniamo l'1 fantasma solo di 7, quello di $10^{-7}$ shifta a destra lasciando a = 0 !!!)

$$\begin{aligned} +10^{-7}{}_{4B} &\cong 0|01100111|10101101011111110010101 \\ &\cong 0|01101000||11010110101111111001010\,(1) \\ &\cong 0|01101001||01101011010111111100101\,(01) \\ &\cdots \\ &\cong 0|01111111||00000000000000000000000\,(11010110101111111110010101) \\ &\cong 0|10000000||00000000000000000000000\,(01101011010111111110010101) \\ &\cong 0|10000001||00000000000000000000000\,(00110101101011111111110010101) \\ &\cong 0|10000010||00000000000000000000000\,(00011010110101111111110010101) \\ &\cong 0|10000011||00000000000000000000000\,(00001101011010111111111110010101) \end{aligned} \tag{2.7}$$

Dovendo effettuare questa operazione per ben 26 volte per colmare la distanza tra 103 e 129, tutte le 23 cifre della mantissa scompariranno oltre la $23^a$ che è l'ultima utile. A questo punto, la somma darà il seguente risultato

$$+\,7._{4B} + 10^{-7}{}_{4B} = +7._{4B} \tag{2.8}$$

Questo risultato è la diretta conseguenza della precisione limitata (a 6 cifre) della rappresentazione a 4B utilizzata. Qualche altro esempio:

$$+\,1._{4B} = +\left(1\cdot 2^0\right) = +2^0\left(1\cdot 2^0\right) = \underline{+}2^{\overline{127}-127} = 0|01111111|00000000000000000000000\,(1) \tag{2.9}$$

| word | m | bias | $r_{N-1}$ | e | e - bias | f | r |
|---|---|---|---|---|---|---|---|
| 4B | 23 | 127 | 0 | 0 | -127 | 0 | +0 |
| | | | 1 | 0 | -127 | 0 | -0 |
| | | | **0, 1** | **0** | **-127** | $\mathbf{1.1920929 \cdot 10^{-7} \div 0.99999988}$ | $\mathbf{\pm 1.4012985 \cdot 10^{-45} \div \pm 1.1754942 \cdot 10^{-38}}$ |
| | | | 0, 1 | 1 ÷ 254 | -126 ÷ 127 | $0 \div 0.99999988$ | $\pm 1.1754944 \cdot 10^{-38} \div \pm 3.4028235 \cdot 10^{38}$ |
| | | | 0 | 255 | 128 | 0 | +Infinity |
| | | | 1 | 255 | 128 | 0 | -Infinity |
| | | | 0 | 255 | 128 | $1.1920929 \cdot 10^{-7} \div 0.99999988$ | (signaling) NaN |
| | | | 1 | 255 | 128 | $1.1920929 \cdot 10^{-7} \div 0.99999988$ | (quiet) NaN |
| 8B | 52 | 1,023 | 0 | 0 | -1,023 | 0 | +0 |
| | | | 1 | 0 | -1,023 | 0 | -0 |
| | | | 0, 1 | 1 ÷ 2,046 | -1,022 ÷ 1,023 | $0 \div 1$ | $\pm 2.2250738585072201 \cdot 10^{-308} \div \pm 1.7976931348623\overline{16} \cdot 10^{308}$ |
| | | | 0 | 2,047 | 1,024 | 0 | +Infinity |
| | | | 1 | 2,047 | 1,024 | 0 | -Infinity |
| | | | 0 | 2,047 | 1,024 | $2.2204460492503\overline{13} \cdot 10^{-16} \div 1$ | (signaling) NaN |
| | | | 1 | 2,047 | 1,024 | $2.2204460492503\overline{13} \cdot 10^{-16} \div 1$ | (quiet) NaN |

Tabella 2.2: Range delle rappresentazioni dei numeri reali

$$\begin{aligned} +17._{4B} &= +\left(1\cdot 2^4 + 1\cdot 2^0\right) = +2^4\left(1\cdot 2^0 + 1\cdot 2^{-4}\right) = \underline{+}2^{\underline{131}-127}\left(1+\underline{1}\cdot 2^{-4}\right) \\ &= 0|10000011|00010000000000000000 \end{aligned} \tag{2.10}$$

$$\begin{aligned} +10^{-6}{}_{4B} &\cong +2^{-43}\cdot 8,796,093 \\ &= +2^{-43}\left(10000110001101111011101\underline{1}\right) \\ &= +2^{-20}\left(1.00001100011011110111\underline{01}\right) \\ &= \underline{+}2^{\underline{107}-127}\left(1+0.\underline{00001100011011110111\underline{01}}\right) \\ &= 0|01101011|00001100011011110111\underline{01} \end{aligned} \tag{2.11}$$

$$\begin{aligned} +10^{-6}{}_{4B} &\cong 0|01101011|00001100011011110111\underline{01} \\ &\cong 0|01101100||10000110001101111011\underline{10}\,(1) \\ &\cong 0|01101101||01000011000110111101\underline{11}\,(01) \\ &\dots \\ &\cong 0|01111111||00000000000000000010\underline{00}\,(01100011011110111\underline{01}) \\ &\cong 0|10000000||00000000000000000001\underline{00}\,(00110001101111011\underline{101}) \\ &\cong 0|10000001||00000000000000000000\underline{10}\,(00011000110111101\underline{111}) \\ &\cong 0|10000010||00000000000000000000\underline{01}\,(00001100011011110\underline{1101}) \\ &\cong 0|10000011||00000000000000000000\underline{00}\,(10000110001101111\underline{0111}\underline{01}) \end{aligned} \tag{2.12}$$

da cui

$$+\,1._{4B} + 10^{-6}{}_{4B} = 0|01111111|00000000000000000010\underline{00} = 1.000001_{4B} \tag{2.13}$$

$$+\,1._{4B} + 10^{-7}{}_{4B} = 0|01111111|00000000000000000000\underline{00} = 1._{4B} \tag{2.14}$$

$$+\,7._{4B} + 10^{-6}{}_{4B} = 0|10000001|11000000000000000000\underline{10} = 7.000001_{4B} \tag{2.15}$$

$$+\,17._{4B} + 10^{-6}{}_{4B} = 0|10000011|00010000000000000000\underline{00} = 17._{4B} \tag{2.16}$$

In conclusione, la precisione limitata della rappresentazione utilizzata porta ad un errore di troncamento che deve essere tenuto sotto controllo per evitare che lo stesso mini la stabilità degli algoritmi numerici utilizzati e/o che renda inutilizzabili i risultati della computazione numerica perché ottenuti con un numero troppo piccolo di cifre significative.

## 2.5 Propagazione dell'errore di troncamento [1, Cap. 3, Sez. 3.3-3.5]

Immaginiamo di avere due numeri reali positivi (a e b) in una qualche rappresentazione a # byte. Li potremmo esprimere come segue

$$a_{\#B} = a\left(1 \pm \varepsilon_a\right) \tag{2.17}$$

dove a è il valore esatto, $a_{\#B}$ è il valore conservato in memoria e $\varepsilon_a \approx \varepsilon_m$ è l'errore relativo di troncamento dovuto alla rappresentazione.

Se sommiamo i due numeri abbiamo

$$\begin{aligned} c_{\#B} &= a_{\#B} + b_{\#B} = a\left(1 \pm \varepsilon_a\right) + b\left(1 \pm \varepsilon_b\right) = c \pm \left(a\varepsilon_a + b\varepsilon_b\right) \\ \frac{c_{\#B}}{c} &= 1 \pm \frac{a\varepsilon_a + b\varepsilon_b}{c} \end{aligned} \tag{2.18}$$

Può succedere che (i) a $\gg$ b $\Rightarrow$ c $\approx$ a, (ii) a $\ll$ b $\Rightarrow$ c $\approx$ b o (iii) a $\approx$ b $\Rightarrow$ c $\approx$ 2a, al che avremmo

$$(i) \qquad \frac{c_{\#B}}{c} \approx 1 \pm \varepsilon_a \tag{2.19}$$

$$(ii) \qquad \frac{c_{\#B}}{c} \approx 1 \pm \varepsilon_b \tag{2.20}$$

$$(iii) \qquad \frac{c_{\#B}}{c} \approx 1 \pm \varepsilon_a \tag{2.21}$$

In questo caso, l'errore relativo di c è dell'ordine di quelli di a e b e quindi di quello di precisione della machina.

Se sottraiamo i due numeri abbiamo

$$\begin{aligned} c_{\#B} &= a_{\#B} - b_{\#B} = a\left(1 \pm \varepsilon_a\right) - b\left(1 \pm \varepsilon_b\right) = c \pm \left(a\varepsilon_a + b\varepsilon_b\right) \\ \frac{c_{\#B}}{c} &= 1 \pm \frac{a\varepsilon_a + b\varepsilon_b}{|c|} \end{aligned} \tag{2.22}$$

Può succedere che (i) a $\gg$ b $\Rightarrow$ c $\approx$ a, (ii) a $\ll$ b $\Rightarrow$ c $\approx$ -b o (iii) a $\approx$ b $\Rightarrow$ c $\approx$ 0, |c| $\ll$ a , al che avremmo

$$(i) \quad \frac{c_{\#B}}{c} \approx 1 \pm \varepsilon_a \tag{2.23}$$

$$(ii) \quad \frac{c_{\#B}}{c} \approx 1 \pm \varepsilon_b \tag{2.24}$$

$$(iii) \quad \frac{c_{\#B}}{c} \approx 1 \pm 2\frac{a}{|c|}\varepsilon_a \tag{2.25}$$

Stavolta, nel caso in cui a $\approx$ b, l'errore relativo su c esplode potendo diventare svariate volte più grande della precisione della macchina ed andando ad inficiare un numero elevato di cifre decimali che perderebbero di significato. Questo è dovuto alla quasi cancellazione della mantissa nell'operazione di sottrazione con una risultante drastica riduzione delle cifre decimali significative nell'operazione di ricostruzione della rappresentazione con a = 1. Un esempio:

$$+1.000001_{4B} \cong 0|01111111|00000000000000000001000 \tag{2.26}$$

$$+1._{4B} \cong 0|01111111|00000000000000000000000 \tag{2.27}$$

$$+1.000001_{4B} - (+1._{4B}) = 0|01101011|00000000000000000000000 = 0.953674 \cdot 10^{-6}{}_{4B} \tag{2.28}$$

In questo esempio, l'errore relativo è dell'ordine del 5% (= 1 - $\frac{0.953674 \cdot 10^{-6}}{1. \cdot 10^{-6}} \approx 0.05$) che è pari a circa $5 \cdot 10^5$ ($\approx 2\frac{1}{1. \cdot 10^{-6}}$) volte l'errore di precisione della macchina (circa $1 \cdot 10^{-7}$) !!!

Se moltiplichiamo o dividiamo i due numeri abbiamo

$$\begin{aligned} c_{\#B} &= a_{\#B} \cdot b_{\#B} = a\,(1 \pm \varepsilon_a) \cdot b\,(1 \pm \varepsilon_b) = c \pm c\,(\varepsilon_a + \varepsilon_b) \\ \frac{c_{\#B}}{c} &= 1 \pm (\varepsilon_a + \varepsilon_b) \end{aligned} \tag{2.29}$$

Ad ogni operazione di moltiplicazione/divisione, l'errore relativo si incrementa di una quantità dell'ordine della precisione della macchina. Ammettendo che gli errori si sommino casualmente, dopo N passi, possiamo aspettarci che l'errore relativo non sia diventato N$\varepsilon_m$, ma soltanto $\sqrt{N}\varepsilon_m$. Ad ogni modo, dopo svariate ore di calcolo su processori capaci di milioni di operazioni al secondo, questo errore può completamente inficiare i risultati dei nostri calcoli.

## 2.6 Tipi di errore e minimizzazione dell'errore totale [1, Cap. 3, Sez. 3.1, 3.9-3.12]

Riassumendo, possiamo incontrare 4 tipi di errore:

1. Errori "umani": errori dovuti alla nostra disattenzione o all'inesattezza della teoria da noi formulata.
2. Errori casuali: errori dovuti all'elettronica, all'alimentazione, ai raggi cosmici, ...
3. Errori di troncamento: errori dovuti alla precisione finita, intrinseca della rappresentazione scelta, con cui possiamo conservare i numeri reali in memoria. Può essere stimato in $\sqrt{N}\varepsilon_m\,(\#B)$.
4. Errori di approssimazione: errori dovuti alle approssimazioni matematiche intrinseche agli algoritmi numerici che utilizziamo (e.g. sostituiamo serie infinite con serie finite, intervalli infinitesimi con intervalli finiti, ...). Può essere spesso modellizato con $\frac{\alpha}{N^\beta}$.

Immaginiamo di avere un algoritmo con un errore di approssimazione al passo N pari a $\frac{\alpha}{N^\beta}$, quale il numero di passi da effettuare per minimizzare l'errore totale

$$\varepsilon\,(N) = \frac{\alpha}{N^\beta} + \sqrt{N}\varepsilon_m \tag{2.30}$$

Effettuando una semplice differenziazione rispetto ad N, abbiamo

$$\frac{d}{dN}\varepsilon\,(N) = -\frac{\alpha\beta}{N^{\beta+1}} + \frac{1}{2}\frac{1}{\sqrt{N}}\varepsilon_m = 0 \tag{2.31}$$

$$\Rightarrow N_{min} = \left(\frac{2\alpha\beta}{\varepsilon_m}\right)^{\frac{1}{\beta+\frac{1}{2}}} \tag{2.32}$$

$$\Rightarrow \varepsilon_{min} = \varepsilon\,(N_{min}) = \alpha\left(\frac{2\alpha\beta}{\varepsilon_m}\right)^{-\frac{\beta}{\beta+\frac{1}{2}}} + \left(\frac{2\alpha\beta}{\varepsilon_m}\right)^{\frac{1}{2\beta+1}}\varepsilon_m \tag{2.33}$$

Si possono studiare vari casi per verificare se è preferibile investire in un algoritmo più veloce ($\beta$ maggiore) o diminuire $\varepsilon_m$, aumentando il numero di byte della rappresentazione. I risultati sono riassunti in Tabella 2.3 e in Figura 2.1.

| | **S**: $\varepsilon_m\,(4B) \simeq 10^{-6}$ | **D**: $\varepsilon_m\,(8B) \simeq 10^{-16}$ |
|---|---|---|
| **A**: $\alpha = 1 \quad \beta = 2$ | $N_{min} = 437 \quad \varepsilon_{min} = 3 \cdot 10^{-5}$ | $N_{min} = 4373448 \quad \varepsilon_{min} = 3 \cdot 10^{-13}$ |
| **B**: $\alpha = 2 \quad \beta = 4$ | $N_{min} = 40 \quad \varepsilon_{min} = 7 \cdot 10^{-6}$ | $N_{min} = 6655 \quad \varepsilon_{min} = 9 \cdot 10^{-15}$ |

Tabella 2.3: Confronto tra efficienza degli algoritmi e precisione della rappresentazione

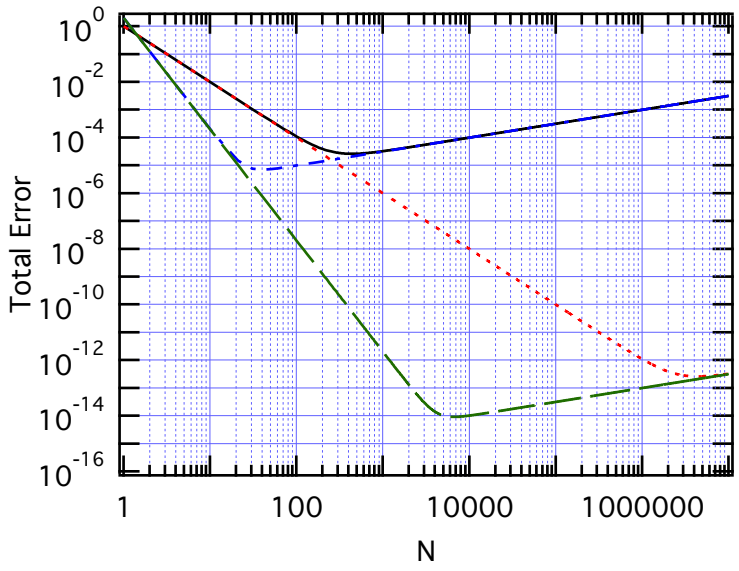


Figura 2.1: Confronto tra efficienza degli algoritmi e precisione della rappresentazione: **AS** linea nera, **AD** linea rossa a punti, **BS** linea blu a punti e tratti, **BD** linea verde a tratti

# Capitolo 3

# Interpolazione di funzioni

Come possiamo interpolare una funzione (i.e. calcolarne il valore in punto generico senza conoscerne la forma analitica) di cui conosciamo il valore solo in un numero finito di punti?

## 3.1 Interpolazione di Lagrange [2, Cap. 2, Sez. 2.1]

Immaginiamo di voler interpolare una funzione incognita $y = f(x)$ di cui sono noti solo i valori $y_i$, con $i = 0, \ldots, N$, in $N+1$ punti $x_i$. In prima approssimazione, il valore della funzione $f(x)$, per ogni punto $x$ compreso tra $x_j$ ed $x_{j+1}$ (i.e. $x_j \leq x \leq x_{j+1}$), si può ottenere tramite un interpolazione lineare e cioè come il valore calcolato per la retta passante per i due punti $x_j$ ed $x_{j+1}$

$$\tilde{f}_1(x) = mx + n \Rightarrow \begin{cases} y_j = mx_j + n \\ y_{j+1} = mx_{j+1} + n \end{cases} \Rightarrow \begin{cases} m = \frac{y_{j+1}-y_j}{x_{j+1}-x_j} \\ n = y_j - \frac{y_{j+1}-y_j}{x_{j+1}-x_j} x_j \end{cases} \tag{3.1}$$

$$\tilde{f}_1(x) = y_j + \frac{y_{j+1}-y_j}{x_{j+1}-x_j}(x - x_j) = \frac{x - x_{j+1}}{x_j - x_{j+1}} y_j + \frac{x - x_j}{x_{j+1} - x_j} y_{j+1} = \sum_{l=j}^{j+1} \frac{x - x_{\bar{l}}}{x_l - x_{\bar{l}}} y_l \tag{3.2}$$

dove $\bar{j} = j+1, \overline{j+1} = j$. Per valutare l'errore commesso usando un'interpolazione lineare possiamo ricorrere alla costruzione di una parabola che passi esattamente per i tre punti $x_j$, $x_{j+1}$ ed $\tilde{x}$ ($x_j < \tilde{x} < x_{j+1}$). Quest'ultimo punto, $\tilde{x}$, possiamo immaginare sia il punto specifico in cui vogliamo conoscere il valore della funzione incognita $f$, $f(\tilde{x})$, tramite la procedura di interpolazione. Questa parabola avrà come espressione

$$g_{\tilde{x}}(x) = \tilde{f}_1(x) + \Delta_{\tilde{x}}(x) \tag{3.3}$$

$$\Delta_{\tilde{x}}(x) = \frac{1}{2} \alpha_{\tilde{x}} (x - x_j)(x - x_{j+1}) \tag{3.4}$$

Così, $\Delta_{\tilde{x}}(x)$ risulta direttamente connesso con la stima dell'errore che volevamo ottenere (i.e. $\Delta_{\tilde{x}}(\tilde{x}) = g_{\tilde{x}}(\tilde{x}) - \tilde{f}_1(\tilde{x}) = f(\tilde{x}) - \tilde{f}_1(\tilde{x})$). Ora, se imponiamo che lo sviluppo in serie di Taylor di $f(x)$ intorno a $\tilde{x}$ al secondo ordine

$$f(x) \cong f(\tilde{x}) + f'(\tilde{x})(x - \tilde{x}) + \frac{1}{2} f''(\tilde{x})(x - \tilde{x})^2 \tag{3.5}$$

passi per $[x_j, y_j]$ e $[x_{j+1}, y_{j+1}]$ e da questo determiniamo $f(\tilde{x})$ e $f'(\tilde{x})$ in funzione di $f''(\tilde{x})$ otteniamo, ovviamente, un'espressione identica a quella di $g_{\tilde{x}}(x)$ dove $\alpha_{\tilde{x}} = f''(\tilde{x})$. Questo ci permette di stimare il massimo errore che è possibile commettere interpolando linearmente per un punto $x$ compreso tra $x_j$ ed $x_{j+1}$ (i.e. $x_j \leq x \leq x_{j+1}$)

$$\frac{d}{dx} \Delta_{\tilde{x}}(x) = 0 \Rightarrow x_{max} = \frac{1}{2}(x_j + x_{j+1}) \tag{3.6}$$

$$\Delta = \frac{1}{2} \left( \max_{x_j < x < x_{j+1}} |f''(x)| \right) |(x_{max} - x_j)(x_{max} - x_{j+1})| = \frac{1}{8} \left( \max_{x_j < x < x_{j+1}} |f''(x)| \right) (x_{j+1} - x_j)^2 \tag{3.7}$$

La formula (3.2) può essere facilmente generalizzata in modo da sfruttare tutti i valori noti della funzione, preservando il vincolo che se $x$ coincide con uno dei punti $x_i$, il valore restituito per la funzione sia proprio $y_i$ (così come accadeva per la semplice interpolazione lineare rispetto ai due punti scelti)

$$\tilde{f}(x) = \sum_{l=0}^{N} p_l(x) y_l \qquad p_l(x) = \prod_{m \neq l} \frac{x - x_m}{x_l - x_m} \tag{3.8}$$

Verifica: $x = x_k \Rightarrow \tilde{f}(x_k) = \sum_{l=0}^{N} p_l(x_k)\, y_l = \sum_{l=0}^{N} \delta_{lk} y_l = y_k$. Questa interpolazione restituisce ovviamente un polinomio di ordine $N$ in $x$. L'errore commesso usando l'interpolazione di ordine $N$ può essere stimato facendo ricorso allo stesso ragionamento che abbiamo utilizzato per l'interpolazione lineare

$$\Delta_{\tilde{x}}(x) = \frac{f^{(N+1)}(\tilde{x})}{(N+1)!}(x-x_0)(x-x_1)\cdots(x-x_N) \tag{3.9}$$

da cui, se $x_i = x_0 + ih$, abbiamo (per $N$ dispari)

$$\frac{d}{dx}\Delta_{\tilde{x}}(x) = 0 \Rightarrow x_{max} = \frac{1}{2}(x_0 + x_N) = x_0 + \frac{N}{2}h \tag{3.10}$$

$$\Delta = \frac{(N!!)^2}{2^{N+1}(N+1)!}\left(\max_{x_0<x<x_N}\left|f^{(N+1)}(x)\right|\right)h^{N+1} \tag{3.11}$$

## 3.2 Metodo di Aitken [2, Cap. 2, Sez. 2.1]

La formula (3.8) può essere riscritta in maniera ricorsiva

$$\begin{aligned}
\tilde{f}(x) &= \sum_{l=0}^{N}\prod_{m\neq l}\frac{x-x_m}{x_l-x_m}y_l = y_{012\cdots N} = \prod_{m\neq 0}\frac{x-x_m}{x_0-x_m}y_0 + \prod_{m\neq N}\frac{x-x_m}{x_N-x_m}y_N \\
&+ \sum_{l=1}^{N-1}\frac{x-x_0}{x_l-x_0}\frac{x-x_N}{x_l-x_N}\prod_{m\neq 0,l,N}\frac{x-x_m}{x_l-x_m}y_l \qquad (3.12)\\
&= \frac{x-x_N}{x_0-x_N}\prod_{m\neq 0,N}\frac{x-x_m}{x_0-x_m}y_0 + \frac{x-x_0}{x_N-x_0}\prod_{m\neq 0,N}\frac{x-x_m}{x_N-x_m}y_N \\
&+ \sum_{l=1}^{N-1}\frac{x-x_0}{x_l-x_0}\frac{x-x_N}{x_l-x_N}\frac{x_0-x_l+x_l-x_N}{x_0-x_N}\prod_{m\neq 0,l,N}\frac{x-x_m}{x_l-x_m}y_l \qquad (3.13)\\
&= \frac{x-x_N}{x_0-x_N}\prod_{m\neq 0,N}\frac{x-x_m}{x_0-x_m}y_0 + \frac{x-x_0}{x_N-x_0}\prod_{m\neq 0,N}\frac{x-x_m}{x_N-x_m}y_N \\
&+ \frac{x-x_0}{x_N-x_0}\sum_{l=1}^{N-1}\frac{x-x_N}{x_l-x_N}\prod_{m\neq 0,l,N}\frac{x-x_m}{x_l-x_m}y_l + \frac{x-x_N}{x_0-x_N}\sum_{l=1}^{N-1}\frac{x-x_0}{x_l-x_0}\prod_{m\neq 0,l,N}\frac{x-x_m}{x_l-x_m}y_l \qquad (3.14)\\
&= \frac{x-x_N}{x_0-x_N}\left(\prod_{m\neq 0,N}\frac{x-x_m}{x_0-x_m}y_0 + \sum_{l=1}^{N-1}\prod_{m\neq l,N}\frac{x-x_m}{x_l-x_m}y_l\right) \\
&+ \frac{x-x_0}{x_N-x_0}\left(\prod_{m\neq 0,N}\frac{x-x_m}{x_N-x_m}y_N + \sum_{l=1}^{N-1}\prod_{m\neq 0,l}\frac{x-x_m}{x_l-x_m}y_l\right) \qquad (3.15)\\
&= \frac{x-x_N}{x_0-x_N}\left(\sum_{l=0}^{N-1}\prod_{m\neq l,N}\frac{x-x_m}{x_l-x_m}y_l\right) + \frac{x-x_0}{x_N-x_0}\left(\sum_{l=1}^{N}\prod_{m\neq 0,l}\frac{x-x_m}{x_l-x_m}y_l\right) \qquad (3.16)\\
&= \frac{x-x_N}{x_0-x_N}y_{012\cdots N-1} + \frac{x-x_0}{x_N-x_0}y_{123\cdots N} \qquad (3.17)
\end{aligned}$$

In generale, abbiamo

$$y_{i\cdots j} = \frac{x-x_j}{x_i-x_j}y_{i\cdots j-1} + \frac{x-x_i}{x_j-x_i}y_{i+1\cdots j} \tag{3.18}$$

dove $y_{ii} = y_i$. Se definiamo $j = i+k+1$ abbiamo

$$y_{i\cdots i+k+1} = \frac{x-x_{i+1+k}}{x_i-x_{i+1+k}}y_{i\cdots i+k} + \frac{x-x_i}{x_{i+1+k}-x_i}y_{i+1\cdots i+1+k} \tag{3.19}$$

e se definiamo $y_i^{(k)} = y_{i\cdots i+k}$ abbiamo

$$y_i^{(k+1)} = \frac{x-x_{i+1+k}}{x_i-x_{i+1+k}}y_i^{(k)} + \frac{x-x_i}{x_{i+1+k}-x_i}y_{i+1}^{(k)} \tag{3.20}$$

dove $y_i^{(0)} = y_i$, $k = 0\ldots N-1$ e $i = 0\ldots N-1-k$. Da cui possiamo calcolare in sequenza ricorsiva e ridefinendo $\widetilde{y}_i = y_i^{(k)}$ ad ogni passo $k$. Il valore interpolato $\tilde{f}(x)$ sarà pari a $y_0^{(N)}$ e cioè a $\widetilde{y}_0$ dopo $N$ passi.

## 3.3 Metodo delle differenze [2, Cap. 2, Sez. 2.1]

Il metodo di Aitken ha un limite di precisione dovuto alla ricerca diretta del valore approssimato tramite approssimazioni successive che cambiano poco il numero. Sarebbe preferibile cercare solo le differenze

$$\Delta^+_{i,j} = y_{i\cdots j} - y_{i+1\cdots j} = \frac{x-x_j}{x_i-x_j} y_{i\cdots j-1} + \left(\frac{x-x_i}{x_j-x_i} - 1\right) y_{i+1\cdots j} \tag{3.21}$$

$$= \frac{x-x_j}{x_i-x_j}\left(y_{i\cdots j-1} - y_{i+1\cdots j}\right) = \frac{x-x_j}{x_i-x_j}\left(y_{i\cdots j-1} - y_{i+1\cdots j-1} + y_{i+1\cdots j-1} - y_{i+1\cdots j}\right) \tag{3.22}$$

$$= \frac{x-x_j}{x_i-x_j}\left(\Delta^+_{i,j-1} - \Delta^-_{i+1,j}\right) \tag{3.23}$$

$$\Delta^-_{i,j} = y_{i\cdots j} - y_{i\cdots j-1} = \frac{x-x_i}{x_i-x_j}\left(\Delta^+_{i,j-1} - \Delta^-_{i+1,j}\right) \tag{3.24}$$

dove $\Delta^\pm_{i,i} = y_i$. Se definiamo $j = i+k+1$ abbiamo

$$\Delta^+_{i,i+k+1} = \frac{x-x_{i+k+1}}{x_i-x_{i+k+1}}\left(\Delta^+_{i,i+k} - \Delta^-_{i+1,i+1+k}\right) \tag{3.25}$$

$$\Delta^-_{i,i+k+1} = \frac{x-x_i}{x_i-x_{i+k+1}}\left(\Delta^+_{i,i+k} - \Delta^-_{i+1,i+1+k}\right) \tag{3.26}$$

e se definiamo $\Delta^{\pm(k)}_i = \Delta^\pm_{i,i+k}$ abbiamo

$$\Delta^{+(k+1)}_i = \frac{x-x_{i+k+1}}{x_i-x_{i+k+1}}\left(\Delta^{+(k)}_i - \Delta^{-(k)}_{i+1}\right) = \left(x-x_{i+k+1}\right) a^{(k)}_i \tag{3.27}$$

$$\Delta^{-(k+1)}_i = \frac{x-x_i}{x_i-x_{i+k+1}}\left(\Delta^{+(k)}_i - \Delta^{-(k)}_{i+1}\right) = \left(x-x_i\right) a^{(k)}_i \tag{3.28}$$

$$a^{(k)}_i = \frac{1}{x_i-x_{i+k+1}}\left(\Delta^{+(k)}_i - \Delta^{-(k)}_{i+1}\right) \tag{3.29}$$

dove $\Delta^{\pm(0)}_i = y_i$, $k = 1\ldots N-1$ e $i = 1\ldots N-k$. Da cui possiamo calcolare in sequenza ricorsiva tutti i $\Delta^{\pm(k)}_i$. Il valore interpolato $\tilde{f}(x)$ sarà dato da

$$\tilde{f}(x) = y_{0\cdots N} = \Delta^{+(N)}_0 + y_{1\cdots N} = \Delta^{+(N)}_0 + \Delta^{-(N-1)}_1 + y_{1\cdots N-1} \tag{3.30}$$

$$= \Delta^{+(N)}_0 + \Delta^{-(N-1)}_1 + \Delta^{+(N-2)}_1 + y_{2\cdots N-1} \tag{3.31}$$

$$= \Delta^{+(N)}_0 + \Delta^{-(N-1)}_1 + \Delta^{+(N-2)}_1 + \Delta^{-(N-3)}_2 + y_{2\cdots N-2} \tag{3.32}$$

$$= \Delta^{+(N)}_0 + \sum_{i=1}^{n}\left(\Delta^{-(N-(2i-1))}_i + \Delta^{+(N-2i)}_i\right) \qquad \text{se } N = 2n \tag{3.33}$$

$$= \Delta^{+(N)}_0 + \sum_{i=1}^{n}\left(\Delta^{-(N-(2i-1))}_i + \Delta^{+(N-2i)}_i\right) + \Delta^{-(0)}_{n+1} \qquad \text{se } N = 2n+1 \tag{3.34}$$

# Capitolo 4

# Derivata numerica di funzioni

Come possiamo derivare una funzione di cui conosciamo il valore solo in un numero finito di punti o di cui non è agevole il calcolo analitico della funzione derivata?

## 4.1 Formule a più punti [2, Cap. 3, Sez. 3.1]

Immaginiamo di conoscere di una funzione incognita $y = f(x)$ solo i valori $y_i$, con $i = 0, \dots, N$, in $N + 1$ punti $x_i = x_0 + ih$.

Possiamo utilizzare la formula di Taylor per esprimere la funzione in uno dei suoi punti

$$f(x_{i\pm n}) = \sum_{l=0}^{\infty} \frac{f^{(l)}(x_i)}{l!} (\pm nh)^l = \sum_{l=0}^{\infty} \frac{f^{(2l)}(x_i)}{2l!} (nh)^{2l} \pm \sum_{l=0}^{\infty} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} (nh)^{2l+1} \tag{4.1}$$

$$f(x_{i+n}) - f(x_{i-n}) = 2 \sum_{l=0}^{\infty} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} (nh)^{2l+1} = 2nhf'(x_i) + 2 \sum_{l=1}^{\infty} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} (nh)^{2l+1} \tag{4.2}$$

$$f'(x_i) = \frac{f(x_{i+n}) - f(x_{i-n})}{2nh} - \sum_{l=1}^{\infty} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} (nh)^{2l} = \frac{f(x_{i+n}) - f(x_{i-n})}{2nh} + O\left(h^2\right) \tag{4.3}$$

$$\sum_{n=1}^{m} \alpha_n f'(x_i) \cong \sum_{n=1}^{m} \alpha_n \frac{f(x_{i+n}) - f(x_{i-n})}{2nh} - \sum_{l=1}^{m-1} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} h^{2l} \sum_{n=1}^{m} \alpha_n n^{2l} \tag{4.4}$$

$$f'(x_i) \cong \frac{\sum_{n=1}^{m} \alpha_n \frac{f(x_{i+n}) - f(x_{i-n})}{2nh}}{\sum_{n=1}^{m} \alpha_n} + O\left(h^{2m}\right) \tag{4.5}$$

Fissato $\alpha_1 = 1$, per $m = 2$ abbiamo $\alpha_2 = -\frac{1}{4}$; per $m = 3$ abbiamo $\alpha_2 = -\frac{2}{5}$ e $\alpha_3 = \frac{1}{15}$.

Possiamo applicare lo stesso procedimento per calcolare la derivata seconda

$$f(x_{i\pm n}) = \sum_{l=0}^{\infty} \frac{f^{(l)}(x_i)}{l!} (\pm nh)^l = \sum_{l=0}^{\infty} \frac{f^{(2l)}(x_i)}{2l!} (nh)^{2l} \pm \sum_{l=0}^{\infty} \frac{f^{(2l+1)}(x_i)}{(2l+1)!} (nh)^{2l+1} \tag{4.6}$$

$$f(x_{i+n}) + f(x_{i-n}) = 2 \sum_{l=0}^{\infty} \frac{f^{(2l)}(x_i)}{2l!} (nh)^{2l} = 2f(x_i) + n^2h^2f''(x_i) + 2 \sum_{l=2}^{\infty} \frac{f^{(2l)}(x_i)}{2l!} (nh)^{2l} \tag{4.7}$$

$$f''(x_i) = \frac{f(x_{i+n}) + f(x_{i-n}) - 2f(x_i)}{n^2h^2} - 2 \sum_{l=2}^{\infty} \frac{f^{(2l)}(x_i)}{2l!} (nh)^{2(l-1)} = \frac{f(x_{i+n}) + f(x_{i-n}) - 2f(x_i)}{n^2h^2} + O\left(h^2\right) \tag{4.8}$$

$$\sum_{n=1}^{m} \alpha_n f''(x_i) \cong \sum_{n=1}^{m} \alpha_n \frac{f(x_{i+n}) + f(x_{i-n}) - 2f(x_i)}{n^2h^2} - 2 \sum_{l=2}^{m} \frac{f^{(2l)}(x_i)}{2l!} h^{2(l-1)} \sum_{n=1}^{m} \alpha_n n^{2(l-1)} \tag{4.9}$$

$$= \sum_{n=1}^{m} \alpha_n \frac{f(x_{i+n}) + f(x_{i-n}) - 2f(x_i)}{n^2h^2} - 2 \sum_{l=1}^{m-1} \frac{f^{(2l+2)}(x_i)}{(2l+2)!} h^{2l} \sum_{n=1}^{m} \alpha_n n^{2l} \tag{4.10}$$

$$f''(x_i) \cong \frac{\sum_{n=1}^{m} \alpha_n \frac{f(x_{i+n}) + f(x_{i-n}) - 2f(x_i)}{n^2h^2}}{\sum_{n=1}^{m} \alpha_n} + O\left(h^{2m}\right) \tag{4.11}$$

Notiamo che essendo la relazione identica nei due casi, abbiamo gli stessi valori di $\alpha_n$ sia per la derivata prima che seconda.

Nei punti di bordo, per mantenere la precisione prescelta, è necessario ottenere i punti mancanti tramite estrapolazione.

## 4.2 Formula ricorsiva [2, Cap. 3, Sez. 3.1]

Se possiamo calcolare il valore della funzione $f(x)$ in un punto qualsiasi, possiamo costruire un metodo ricorsivo di calcolo della derivata che permette di verificare la precisione raggiunta

$$D(h) = \frac{f(x+h) - f(x-h)}{2h} \tag{4.12}$$

$$D\left(\frac{h}{2}\right) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} \tag{4.13}$$

$$f'(x) \cong \frac{4}{3}D\left(\frac{h}{2}\right) - \frac{1}{3}D(h) \tag{4.14}$$

Dobbiamo continuare a dividere $h$ a metà finche l'errore relativo non è inferiore ad una precisione $\varepsilon$ prescelta: $\left|D\left(\frac{h}{2}\right) - D(h)\right| \leq \varepsilon \left|D\left(\frac{h}{2}\right)\right|$.

# Capitolo 5

# Integrazione numerica di funzioni

Come possiamo integrare una funzione di cui conosciamo il valore solo in un numero finito di punti o di cui non è nota o non è agevole da calcolare la primitiva?

## 5.1 Integrazione per interpolazione [2, Cap. 3, Sez. 3.2]

Immaginiamo di conoscere di una funzione incognita $y = f(x)$ solo i valori $y_i$, con $i = 0, \ldots, N$, in $N + 1$ punti $x_i = x_0 + ih$. Volendo calcolare il seguente integrale

$$S = \int_{x_0}^{x_N} f(x)\, dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)\, dx \tag{5.1}$$

possiamo interpolare la funzione $f(x)$ in $[x_i, x_{i+1}]$ con una funzione costante di valore $y_i$ o $y_{i+1}$ ed ottenere

$$S \approx h \sum_{i=0}^{N-1} y_i \approx h \sum_{i=0}^{N-1} y_{i+1} \tag{5.2}$$

che equivale a ricoprire l'area con rettangoloidi.

Possiamo ottenere un risultato migliore interpolando linearmente nello stesso intervallo

$$f(x) \approx mx + n \Rightarrow \int_{x_i}^{x_{i+1}} f(x)\, dx \approx \frac{1}{2} m x^2 + nx \Big|_{x_i}^{x_{i+1}} = \frac{1}{2} m \left(x_{i+1}^2 - x_i^2\right) + n\left(x_{i+1} - x_i\right) \tag{5.3}$$

$$= \frac{1}{2} h \left[m\left(x_{i+1} + x_i\right) + 2n\right] = \frac{1}{2} h \left(y_{i+1} + y_i\right) \tag{5.4}$$

da cui

$$S \approx \frac{1}{2} h \sum_{i=0}^{N-1} \left(y_i + y_{i+1}\right) = h \sum_{i=0}^{N} w_i y_i + O\left(h^2\right) \qquad w_i = \begin{cases} \frac{1}{2} & i = 0, N \\ 1 & 0 < i < N \end{cases} \tag{5.5}$$

che equivale a mediare i due risultati ottenuti precedentemente o a ricoprire l'aerea con trapezi.

Possiamo ottenere un risultato ancora migliore interpolando parabolicamente in un intervallo doppio $[x_i, x_{i+2}]$ con la richiesta che $N$ sia un numero pari così da avere un numero dispari di punti totali e pari di intervalli

$$f(x) \approx ax^2 + bx + c \Rightarrow \int_{x_i}^{x_{i+2}} f(x)\, dx \approx \frac{1}{3} a x^3 + \frac{1}{2} b x^2 + cx \Big|_{x_i}^{x_{i+2}} \tag{5.6}$$

$$= \frac{1}{3} a \left(x_{i+2}^3 - x_i^3\right) + \frac{1}{2} b \left(x_{i+2}^2 - x_i^2\right) + c\left(x_{i+2} - x_i\right) \tag{5.7}$$

$$= \frac{1}{3} h \left(2a\left(x_{i+2}^2 + x_{i+2} x_i + x_i^2\right) + 3b\left(x_{i+2} + x_i\right) + 6c\right) \tag{5.8}$$

$$= \frac{1}{3} h \left(a\left(x_{i+2}^2 + \left(x_{i+2} + x_i\right)^2 + x_i^2\right) + b\left(x_{i+2} + 2\left(x_{i+2} + x_i\right) + x_i\right) + 6c\right) \tag{5.9}$$

$$= \frac{1}{3} h \left(a\left(x_{i+2}^2 + 4x_{i+1}^2 + x_i^2\right) + b\left(x_{i+2} + 4x_{i+1} + x_i\right) + 6c\right) = \frac{1}{3} h \left(y_{i+2} + 4y_{i+1} + y_i\right) \tag{5.10}$$

da cui

$$S \approx \frac{1}{3} h \sum_{i=0}^{N/2-1} \left(y_{2i} + 4y_{2i+1} + y_{2i+2}\right) = h \sum_{i=0}^{N} w_i y_i + O\left(h^4\right) \qquad w_i = \begin{cases} \frac{1}{3} & i = 0, N \\ \frac{4}{3} & i \neq 2\left(\frac{i}{2}\right) \\ \frac{2}{3} & i = 2\left(\frac{i}{2}\right) \end{cases} \tag{5.11}$$

Questa è la formula di Simpson.

## 5.2 Formula ricorsiva [2, Cap. 3, Sez. 3.2]

Se possiamo calcolare il valore della funzione $f(x)$ in un punto qualsiasi, possiamo costruire un metodo ricorsivo di calcolo degli integrali che permette di verificare la precisione raggiunta

$$S = \int_a^b f(x)\,dx = \int_a^b \sum_{k=0}^{\infty} \frac{(x-a)^k}{k!} f^{(k)}(a)\,dx = \sum_{k=0}^{\infty} \frac{(b-a)^{k+1}}{(k+1)!} f^{(k)}(a) \tag{5.12}$$

Le formule che abbiamo ricavato finora danno invece

$$S_{N+1} = \frac{b-a}{N} \sum_{i=0}^{N} w_i y_i \tag{5.13}$$

Per i trapezi abbiamo

$$S_2^t = \frac{b-a}{2}\left(f(a) + f(b)\right) = \frac{b-a}{2}\left(f(a) + \sum_{k=0}^{\infty} \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.14}$$

da cui

$$\Delta_2^t \equiv S - S_2^t = \sum_{k=0}^{\infty} \frac{(b-a)^{k+1}}{(k+1)!} f^{(k)}(a) - \frac{b-a}{2}\left(f(a) + \sum_{k=0}^{\infty} \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.15}$$

$$\approx -\frac{(b-a)^3}{12} f''(a) \tag{5.16}$$

Se dividiamo l'intervallo in due parti $c = \frac{a+b}{2}$, passando a 3 punti, abbiamo

$$S_3^t = \frac{c-a}{2}\left(f(a) + f(c)\right) + \frac{b-c}{2}\left(f(c) + f(b)\right) = \frac{b-a}{4}\left(f(a) + 2f(c) + f(b)\right) = \frac{1}{2}S_2^t + \frac{b-a}{2} f(c) \tag{5.17}$$

$$= \frac{b-a}{4}\left(f(a) + 2\sum_{k=0}^{\infty} \frac{(c-a)^k}{k!} f^{(k)}(a) + \sum_{k=0}^{\infty} \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.18}$$

$$= \frac{b-a}{4}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{k-1}} + 1\right) \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.19}$$

da cui

$$\Delta_3^t \equiv S - S_3^t = \sum_{k=0}^{\infty} \frac{(b-a)^{k+1}}{(k+1)!} f^{(k)}(a) - \frac{b-a}{4}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{k-1}} + 1\right) \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.20}$$

$$\approx -\frac{(b-a)^3}{48} f''(a) \tag{5.21}$$

In conclusione, abbiamo

$$\left|S_3^t - S_2^t\right| \approx \left|-\frac{(b-a)^3}{16} f''(a)\right| = 3\left|\Delta_3^t\right|$$

da cui

$$\left|\Delta_3^t\right| \approx \frac{1}{3}\left|S_3^t - S_2^t\right| \leq \epsilon$$

per cui, dobbiamo continuare a dividere gli intervalli a metà finche la differenza tra due iterazioni successive non è inferiore a 3 (tre) volte una precisione $\varepsilon$ prescelta.

Per Simpson abbiamo

$$S_3^s = \frac{b-a}{6}\left(f(a) + 4f(c) + f(b)\right) = \frac{b-a}{6}\left(f(a) + 4\sum_{k=0}^{\infty} \frac{(c-a)^k}{k!} f^{(k)}(a) + \sum_{k=0}^{\infty} \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.22}$$

$$= \frac{b-a}{6}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{k-2}} + 1\right) \frac{(b-a)^k}{k!} f^{(k)}(a)\right) \tag{5.23}$$

da cui

$$\Delta_3^s \equiv S - S_3^s = \sum_{k=0}^{\infty} \frac{(b-a)^{k+1}}{(k+1)!} f^{(k)}(a) - \frac{b-a}{6}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{k-2}}+1\right)\frac{(b-a)^k}{k!}f^{(k)}(a)\right) \tag{5.24}$$

$$\approx -\frac{(b-a)^5}{2880} f^{(4)}(a) \tag{5.25}$$

Se dividiamo ogni sotto intervallo in due parti $d = \frac{a+c}{2}$ e $e = \frac{c+b}{2}$, passando a 5 punti, abbiamo

$$S_5^s = \frac{c-a}{6}\left(f(a) + 4f(d) + f(c)\right) + \frac{b-c}{6}\left(f(c) + 4f(e) + f(b)\right) \tag{5.26}$$

$$= \frac{b-a}{12}\left(f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)\right) \tag{5.27}$$

$$= \frac{b-a}{12}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{2k-2}} + \frac{1}{2^{k-1}} + \frac{3^k}{2^{2k-2}} + 1\right)\frac{(b-a)^k}{k!}f^{(k)}(a)\right) \tag{5.28}$$

da cui

$$\Delta_5^s \equiv S - S_5^s = \sum_{k=0}^{\infty} \frac{(b-a)^{k+1}}{(k+1)!} f^{(k)}(a) - \frac{b-a}{12}\left(f(a) + \sum_{k=0}^{\infty}\left(\frac{1}{2^{2k-2}} + \frac{1}{2^{k-1}} + \frac{3^k}{2^{2k-2}} + 1\right)\frac{(b-a)^k}{k!}f^{(k)}(a)\right) \tag{5.29}$$

$$\approx -\frac{1}{16}\frac{(b-a)^5}{2880} f^{(4)}(a) \tag{5.30}$$

In conclusione, abbiamo

$$|S_5^s - S_3^s| \approx \left| -\frac{15}{16}\frac{(b-a)^5}{2880} f^{(4)}(a) \right| = 15\,|\Delta_5^s|$$

da cui

$$|\Delta_5^s| \approx \frac{1}{15}|S_5^s - S_3^s| \leq \epsilon$$

per cui, dobbiamo continuare a dividere gli intervalli a metà finche la differenza tra due iterazioni successive non è inferiore a 15 (quindici) volte una precisione $\varepsilon$ prescelta.

Per costruire la ricorsione dobbiamo tenere in conto che per i trapezi abbiamo (da cui possiamo costruire l'integrale generale aggiungendo punti per ricorsione e fino al raggiungimento di una precisione prescelta)

$$\begin{aligned}
N_1 &= 2 \\
\delta_1 &= (b-a) \\
S_{N_1} = S_2 &= \delta_1\left[\frac{1}{2}f(a) + \frac{1}{2}f(b)\right] \\
N_m &= 2N_{m-1} - 1 \qquad m > 1 \\
\delta_m &= \frac{1}{2}\delta_{m-1} \\
\delta S_m &= \delta_m \sum_{i=1}^{\frac{N_m-1}{2}} f(a + (2i-1)\delta_m) \\
S_{N_m} &= \frac{1}{2}S_{N_{m-1}} + \delta S_m \\
|\Delta_m| = |S - S_{N_m}| &\approx \frac{1}{3}\left|S_{N_m} - S_{N_{m-1}}\right| = \frac{1}{3}\left|\delta S_m - \frac{1}{2}S_{N_{m-1}}\right| \leq \varepsilon
\end{aligned}$$

Nel caso della formula di Simpson, non possiamo evitare di ricalcolare la funzione in tutti i punti, ma potremmo, in principio, utilizzare comunque una parte dell'algoritmo per determinare il numero minimo di punti per raggiungere la precisione prescelta

$$\begin{aligned}
N_1 &= 3 \\
S_{N_1} &= S_3^s \\
N_m &= 2N_{m-1} - 1 \qquad m > 1 \\
S_{N_m} &= S_{N_m}^s \\
|\Delta_m| = |S - S_{N_m}| &\approx \frac{1}{15}\left|S_{N_m} - S_{N_{m-1}}\right| \leq \varepsilon
\end{aligned}$$

Inoltre, possiamo costruire un algoritmo ricorsivo adattativo, utilizzando che

$$
\begin{aligned}
S &= \int_a^b f(x)\,dx = S(a,b) \\
A &= \begin{cases} S_2^t &= \frac{b-a}{2}\left(f(a)+f(b)\right) \\ S_3^s &= \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right) \end{cases} \\
B &= \begin{cases} S_3^t &= \frac{b-a}{4}\left(f(a)+2f\left(\frac{a+b}{2}\right)+f(b)\right) \\ S_5^s &= \frac{b-a}{12}\left(f(a)+4f\left(\frac{3a+b}{4}\right)+2f\left(\frac{a+b}{2}\right)+4f\left(\frac{a+3b}{4}\right)+f(b)\right) \end{cases} \\
\alpha &= \begin{cases} \alpha^t &= 3 \\ \alpha^s &= 15 \end{cases} \\
\Delta &= |S(a,b)-B| \approx \frac{1}{\alpha}|B-A| \\
&\begin{cases} S(a,b) \approx B & \frac{\Delta}{|B|} \leq \varepsilon \\ S(a,b) = S\left(a,\frac{a+b}{2}\right)+S\left(\frac{a+b}{2},b\right) & \frac{\Delta}{|B|} > \varepsilon \end{cases}
\end{aligned}
$$

dove $a_0$ e $b_0$ sono i valori iniziali di $a$ e $b$. Se passiamo alla funzione che calcola l'integrale $a$, $b$, $f$, $f(a)$, $f(b)$ per i trapezi ed anche $f\left(\frac{a+b}{2}\right)$ per Simpson, possiamo evitare il ricalcolo della funzione e questo algoritmo diventa il più competitivo tra tutti.

# Capitolo 6

# Matrici

Come operare numericamente su di una matrice per risolvere un sistema di equazioni lineari, calcolare la sua matrice inversa, il suo determinante ed i suoi autovalori-autovettori?

## 6.1 Sistema di equazioni lineari

Un sistema di equazioni lineari può essere riscritto in maniera matriciale come

$$A \cdot x = b \tag{6.1}$$

dove $A$ è una matrice quadrata $n \times n$, $x$ è il vettore incognito con $n$ elementi e $b$ è il vettore dei termini noti, anch'essi in numero di $n$. Il metodo di eliminazione di Gauss permette di risolvere il sistema riducendo la matrice $A$ ad una matrice triangolare superiore ed eseguendo semplici sostituzioni.

Per ridurre la matrice $A$ alla forma triangolare superiore applichiamo in maniera ricorsiva la seguente formula

$$A_{ij}^{(k)} = A_{ij}^{(k-1)} - \frac{A_{ik}^{(k-1)}}{A_{kk}^{(k-1)}} A_{kj}^{(k-1)} \tag{6.2}$$

dove $A^{(0)} = A$, $k = 1, \ldots, n-1$, $i = k+1, \ldots, n$ e $j = k, \ldots, n$. Anche il termine noto va modificato in maniera ricorsiva

$$b_i^{(k)} = b_i^{(k-1)} - \frac{A_{ik}^{(k-1)}}{A_{kk}^{(k-1)}} b_k^{(k-1)} \tag{6.3}$$

dove $b^{(0)} = b$. Il sistema di equazioni, al passo $n-1$, si sarà ridotto a

$$\sum_{j=i}^{n} A_{ij}^{(n-1)} x_j = b_i^{(n-1)} \tag{6.4}$$

dove $i = 1, \ldots, n$. Quest'ultimo può essere facilmente risolto per sostituzione partendo da $x_n = \frac{b_n^{(n-1)}}{A_{nn}^{(n-1)}}$ e risalendo fino a $x_1$

$$x_i = \frac{1}{A_{ii}^{(n-1)}} \left( b_i^{(n-1)} - \sum_{j=i+1}^{n} A_{ij}^{(n-1)} x_j \right) \tag{6.5}$$

dove $i = n-1, \ldots, 1$.

Possiamo evitare di dividere per numeri piccoli cercando la riga, normalizzata, con il valore più grande del termine di colonna che ci serve ad ogni passo e tener traccia di quest'ultimo tramite un vettore indice $v(i)$. Inizializziamo $v(i) = i$ dove $i = 1, \ldots, n$. Poi, normalizziamo tutte le righe di una matrice ausiliaria $B$ dividendo i loro termini per il loro valore assoluto massimo e cerchiamo nella colonna che vogliamo eliminare il termine

con il valore assoluto massimo ($p = 1$, tiene memoria del numero di permutazioni tra le righe)

$$\begin{aligned} B &= A^{(k-1)} && (6.6)\\ m_i &= \max_j (|B_{ij}|)\ \forall i && (6.7)\\ B_{ij} &= \frac{|B_{ij}|}{m_i}\ \forall i,j && (6.8)\\ q &= \max_{i\geq k} \operatorname{loc}(B_{ik}) && (6.9)\\ r &= v(q) && (6.10)\\ v(q) &= v(k) && (6.11)\\ v(k) &= r && (6.12)\\ \text{if} \quad q \neq k \Rightarrow \quad p &= -p && (6.13) \end{aligned}$$

Le formule precedenti si modificheranno come segue

$$A^{(k)}_{v(i)j} = A^{(k-1)}_{v(i)j} - \frac{A^{(k-1)}_{v(i)k}}{A^{(k-1)}_{v(k)k}} A^{(k-1)}_{v(k)j} \tag{6.14}$$

$$b^{(k)}_{v(i)} = b^{(k-1)}_{v(i)} - \frac{A^{(k-1)}_{v(i)k}}{A^{(k-1)}_{v(k)k}} b^{(k-1)}_{v(k)} \tag{6.15}$$

ed alla fine le $x$ saranno determinate come segue $x_n = \frac{b^{(n-1)}_{v(n)}}{A^{(n-1)}_{v(n)n}}$ e

$$x_i = \frac{1}{A^{(n-1)}_{v(i)i}} \left( b^{(n-1)}_{v(i)} - \sum_{j=i+1}^{n} A^{(n-1)}_{v(i)j} x_j \right) \tag{6.16}$$

dove $i = n-1, \ldots, 1$.

## 6.2 Determinante

Il determinante di $A$ sarà dato da

$$\det A = p \prod_{i=1}^{n} A^{(n-1)}_{v(i)i} \tag{6.17}$$

## 6.3 Inversa

L'inversa di $A$ sarà data dalla seguente formula

$$A \cdot A^{-1} = \mathbf{1} \tag{6.18}$$

ridotta a

$$A \cdot X = B \tag{6.19}$$

per cui $A^{-1}_{nj} = \frac{B^{(n-1)}_{v(n)j}}{A^{(n-1)}_{v(n)n}}$ e

$$A^{-1}_{ij} = \frac{1}{A^{(n-1)}_{v(i)i}} \left( B^{(n-1)}_{v(i)j} - \sum_{k=i+1}^{n} A^{(n-1)}_{v(i)k} A^{-1}_{kj} \right) \tag{6.20}$$

dove $i = n-1, \ldots, 1$ e $j = 1, \ldots, n$. $B^{(0)}_{ij} = \delta_{ij}$ e

$$B^{(k)}_{v(i)j} = B^{(k-1)}_{v(i)j} - \frac{A^{(k-1)}_{v(i)k}}{A^{(k-1)}_{v(k)k}} B^{(k-1)}_{v(k)j} \tag{6.21}$$

dove $k = 1, \ldots, n-1$ e $i = k+1, \ldots, n$.

## 6.4 Autovalori ed Autovettori

Se la matrice A, reale simmetrica o hermitiana, ha l'autovalore di valore assoluto più grande non degenere $\lambda_1$, possiamo determinarlo grazie al metodo iterativo delle potenze. Prendiamo un vettore $v^{(0)}$ random ed applichiamo ad esso ripetutamente la matrice A e normalizziamo il risultato ad ogni passaggio

$$v^{(i)} = \frac{A \cdot v^{(i-1)}}{\left\| A \cdot v^{(i-1)} \right\|} \qquad i > 0 \tag{6.22}$$

In principio, possiamo espandere il vettore $v^{(0)}$ nella base costituita dagli autovettori ortonormali $x_k$, di autovalore $\lambda_k$, di $A$

$$v^{(0)} = \sum_k c_k x_k \tag{6.23}$$

da cui

$$v^{(i)} = \frac{\sum_k c_k \lambda_k^i x_k}{\sqrt{\sum_k \left(c_k \lambda_k^i\right)^2}} = \frac{\lambda_1^i \sum_k c_k \left(\frac{\lambda_k}{\lambda_1}\right)^i x_k}{\left|\lambda_1^i\right| \sqrt{\sum_k c_k^2 \left(\frac{\lambda_k}{\lambda_1}\right)^{2i}}} \cong x_1 \tag{6.24}$$

In conclusione, possiamo ottenere l'autovettore $x_1$ da

$$x_1 = \lim_{i\to\infty} v^{(i)} = \lim_{i\to\infty} \frac{A \cdot v^{(i-1)}}{\left\| A \cdot v^{(i-1)} \right\|} = \lim_{i\to\infty} \frac{A^i \cdot v^{(0)}}{\left\| A^i \cdot v^{(0)} \right\|} \tag{6.25}$$

ed il corrispondente autovalore da

$$\lambda_1 = \frac{x_1^T \cdot A \cdot x_1}{x_1^T \cdot x_1} \tag{6.26}$$

Per ottenere il secondo autovalore di valore assoluto più grande non degenere $\lambda_2$, possiamo ripetere la stessa operazione partendo da un vettore random $w^{(0)}$ cui è stata sottratta la componente rispetto all'autovettore $x_1$: $\tilde{w}^{(0)} = w^{(0)} - \left(x_1^T \cdot w^{(0)}\right) x_1$. La procedura può essere ripetuta finché gli autovalori di valore assoluto più grande sono non degeneri, ma risulta estremamente instabile numericamente. Per ottenere l'autovalore di valore assoluto più piccolo non degenere $\lambda_n$, possiamo ripetere la stessa operazione utilizzando la matrice $A^{-1}$. Per ottenere l'autovalore più vicino ad un numero dato $\mu$, possiamo ripetere la stessa operazione utilizzando la matrice ausiliaria $(A - \mu I)^{-1}$. Come facciamo a scegliere utilmente $\mu$?

Il Teorema di Gershgorin ci assicura che un autovalore $\lambda$ di una matrice $A$ non può *distare* da un elemento della diagonale della matrice $A_{ii}$ (dove $i$ è fissato ricercando la componente $x_i$ di valore assoluto massimo dell'autovettore corrispondente $x$: $A \cdot x = \lambda x$, $x = \{x_n\}$, $i \left| |x_i| = \max_n |x_n| \right.$) più della somma dei valori assoluti di tutti gli altri elementi appartenenti alla stessa riga o colonna (e quindi non più del minimo delle due somme):

$$A \cdot x = \lambda x \tag{6.27}$$

$$\sum_j A_{ij} x_j = \lambda x_i \tag{6.28}$$

$$\sum_{j\neq i} A_{ij} x_j = (\lambda - A_{ii})\, x_i \tag{6.29}$$

$$\frac{\sum_{j\neq i} A_{ij} x_j}{x_i} = \lambda - A_{ii} \tag{6.30}$$

$$|\lambda - A_{ii}| = \left| \frac{\sum_{j\neq i} A_{ij} x_j}{x_i} \right| \leq \sum_{j\neq i} \left| \frac{A_{ij} x_j}{x_i} \right| \leq \sum_{j\neq i} |A_{ij}| \tag{6.31}$$

$$|\lambda - A_{ii}| \leq \min\left( \sum_{j\neq i} |A_{ij}|, \sum_{j\neq i} |A_{ji}| \right)$$

Da ciò è possibile dimostrare che se l'intervallo così costruito intorno ad un elemento della diagonale $A_{ii}$ è disgiunto da tutti gli altri intervalli costruiti intorno agli altri elementi della diagonale, l'intervallo intorno ad $A_{ii}$ contiene un autovalore della matrice $A$. Da ciò, è evidente che $A_{ii}$ è un ottimo candidato per $\mu$. Se $k$ intervalli non risultassero disgiunti, si può dimostrare che nella loro unione sono presenti $k$ autovalori. In questo caso, l'uso delle $A_{ii}$ corrispondenti può portare a convergenze molto lente. Ad ogni modo, uno studio preliminare degli intervalli di Gershgorin può essere di grande aiuto nella ricerca di autovalori ed autovettori tramite il metodo iterativo delle potenze applicato a $(A - \mu I)^{-1}$.

In ultimo, va ricordato che traccia e determinante di una matrice non cambiano sotto operazioni di similarità quali quelle utili per diagonalizzarla e che quindi si possono usare anche queste informazioni per calcolare gli autovalori di una matrice quando quelli incogniti rimasti sono due o uno. Dato un autovalore $\lambda$, l'autovettore corrispondente, va calcolato utilizzando $(A - \mu I)^{-1}$ con $\mu = \lambda \pm \epsilon$ dove il segno ed il valore di $\epsilon$ vanno scelti con estrema cautela in maniera da porsi quanto più vicino possibile a $\lambda$, ma lontano dagli altri autovalori. In particolare, va scelto il segno in maniera tale da mettersi sul lato opposto, rispetto a $\lambda$, del più vicino autovalore (e.g. $\lambda_1 = 1.9$ noto, $\lambda_2 = 2$ noto, ma senza autovettore $\rightarrow \mu = \lambda_2 + 0.1$). Ancora una volta, uno studio preliminare degli intervalli di Gershgorin può essere di grande aiuto.

# Parte II

# Fortran

# Capitolo 7

# Anatomia

## 7.1 Programma

```
PROGRAM nome_programma

! Commento

   USE nome_module
   IMPLICIT NONE
   {Dichiarazioni}

   {Esecuzioni} ! Commento
    #1 = nome_funzione(#2,#3,...)
    CALL nome_subroutine(#1,#2,...)

END PROGRAM nome_programma
```

## 7.2 Module

```
MODULE nome_module

! Commento

   IMPLICIT NONE
   {Dichiarazioni}

CONTAINS

   {Funzioni e Subroutine}

END MODULE nome_module
```

## 7.3 Funzione

```
FUNCTION nome_funzione(#1,#2,...) RESULT(#3)

! Commento

   IMPLICIT NONE
   {Dichiarazioni}

    {Esecuzioni}
    #3 = ... ! obbligatoria

END FUNCTION nome_funzione
```

## 7.4 Subroutine

```
SUBROUTINE nome_subroutine(#1,#2,...)

! Commento

   IMPLICIT NONE
   {Dichiarazioni}

    {Esecuzioni}

END SUBROUTINE nome_subroutine
```

# Capitolo 8

# Tipi

## 8.1 Tipi definiti

**REAL**(#1), Proprietà :: r [= #4]

**INTEGER**(#2), Proprietà :: i [= #5]

**LOGICAL**, Proprietà :: l [= #6]

**CHARACTER**(**LEN**=#3), Proprietà :: s [= #7]

dove #1, #2, #3 sono interi che indicano rispettivamente la rappresentazione reale scelta per r, la rappresentazione intera scelta per i, il numero di caratteri della stringa s.
Proprietà indica un elenco separato da virgole di proprietà tra cui: **Parameter** (che indica che l'oggetto è una costante: è necessario inizializzarlo!!!), **Save** (solo in funzioni o subroutine: indica che l'oggetto mantiene il suo valore tra le diverse chiamate alla funzione o subroutine), ...
[...] indica l'inizializzazione opzionale (tranne che per quelli definiti costanti tramite **Parameter**) dell'oggetto.

## 8.2 Costanti numeriche

#1E#2

#1D#2

#1E#2_#3

dove #1 è un numero reale, #2 è la potenza di dieci in notazione scientifica, #3 è la rappresentazione reale scelta. E (in assenza di #3) corrisponde ad una rappresentazione reale a 4 Byte, D corrisponde ad una rappresentazione reale a 8 Byte.

## 8.3 Vettori, Matrici e Tensori

**REAL**(#1), Proprietà :: r(#2:#3,#4:#5,...)

**INTEGER**(#6), Proprietà :: i(#7:#8,#9:#10,...)

#1 è la rappresentazione reale scelta, #5 è la rappresentazione intera scelta. #2 e #7 sono gli indici minimi lungo la prima dimensione, #3 e #8 sono gli indici massimi lungo la prima dimensione, ... degli oggetti r e i rispettivamente. Il default per gli indici minimi è 1.

# Capitolo 9

# Operazioni

## 9.1 Numeriche

```
#1+#2        ! somma

#1-#2        ! differenza

#1*#2        ! prodotto

#1/#2        ! rapporto

#1**#2       ! elevazione a potenza

LOG(#1)      ! logaritmo in base naturale

LOG10(#1)    ! logaritmo in base 10

REAL(#1,#2)  ! Trasforma l'intero #1 in un reale di rappresentazione #2

INT(#1,#2)   ! Trasforma il reale #1 in un intero di rappresentazione #2
```

Il Fortran esegue un upgrade dei numeri utilizzati alla rappresentazione di ordine superiore tra i due numeri. !!! Il rapporto tra due interi rimane un intero e non ha resto: $1/1 = 1$, ma $1/2 = 0$ !!!.

# Capitolo 10

# Costrutti

## 10.1 If ... Then ... Else If ... Else

```
IF (#1) THEN
      ...
   ELSE IF (#2) THEN
      ...
   ELSE
      ...
END IF
```

dove #1 e #2 sono oggetti di tipo **Logical** che si possono ottenere confrontando, ad esempio, oggetti reali ed interi tramite le particelle: == (uguale), /= (diverso), > (maggiore), < (minore), >= (maggiore o uguale), <= (minore o uguale).

## 10.2 Ciclo Do

```
DO i=inizio,fine,step
   ...
   IF (#1) EXIT
   ...
   IF (#2) CYCLE
   ...
END DO
```

dove i è un oggetto intero che assumerà iterazione per iterazione del ciclo valori compresi tra inizio e fine con passo pari a step; #1 e #2 sono oggetti di tipo **Logical**; **Exit** interrompe il ciclo do ed esce da esso; **Cycle** interrompe l'iterazione e salta a quella successiva. Il valore di default di step è 1.

## 10.3 Ciclo Do While

```
DO WHILE(#1)
   ...
   IF (#2) EXIT
   ...
   IF (#3) CYCLE
   ...
END DO
```

#1, #2 e #3 sono oggetti di tipo **Logical**; **Exit** interrompe il ciclo do ed esce da esso; **Cycle** interrompe l'iterazione e salta a quella successiva.

# Capitolo 11

# Input/Output

## 11.1 Schermo

```
PRINT *,#1,#2,...
WRITE(*,*) #1,#2,...
```

dove #1, #2,... è un elenco di oggetti che verranno stampati a video nel loro formato standard.

## 11.2 File dati

```
OPEN(#1,File=#2)
   ...
   WRITE(#1,*) #3,#4,...
   ...
CLOSE(#1)
```

dove #1 è un intero, #2 è una stringa contente posizione e nome del file, #3, #4,... è un elenco di oggetti che verranno scritti nel file nel loro formato standard seguiti da un accapo.

# Parte III

# Linea di comando (Shell, Compilatore, ...)

## 11.3 Shell

**cd** *#1*
*./#2*
**pwd**

Cambia directory in `#1` o usando `..` al posto di `#1` si torna alla directory precedente nell'albero. Esegue il file eseguibile di nome `#2`. **pwd** mostra la directory corrente.

## 11.4 Compilatore

```
gfrotran #1.f90 -o #1.exe
gfrotran library.f90 #1.f90 -o #1.exe
```

dove `#1` è il nome del file di programma senza estensione.

# Parte IV

# Gnuplot

## 11.5 Grafico

```
gnuplot
> plot [#11:#12] [#13:#14] #1 using #2:#3 with #4 title '#5', #6 using #7:#8
    with #9 title '#10', ...
```

dove #1 e #6 sono i nomi dei file dati racchiusi tra apici (se si usano solo due apici al posto del nome del file viene utilizzato il file precedente: e.g. #6=" allora #6=#1 ), #2 e #7 sono le colonne da usare quali x, #3 e #8 le colonne da usare quali y, #4 e #9 il tipo di stile grafico da usare tra **line**, **linepoints**, **points**, **dots**,..., #5 e #10 sono i nomi da usare nella legenda. Al posto del numero di colonna si può utilizzare una qualsiasi funzione matematica racchiusa tra parentesi dove compaiono i numeri di colonna preceduti dal simbolo del dollaro: e.g. #2=($1/Log($2)), #3=(sqrt($1*$2)). #11 è il minimo lungo l'asse x (può essere omesso ed il valore è determinato dai dati), #12 è il massimo lungo l'asse x (può essere omesso ed il valore è determinato dai dati), #13 è il minimo lungo l'asse y (può essere omesso ed il valore è determinato dai dati), #14 è il massimo lungo l'asse y (può essere omesso ed il valore è determinato dai dati).

## 11.6 Output verso un file

```
> set term pdfcairo
> set output '#1.pdf'
> plot ...
> set output
> set term wxt
```

dove #1 è il nome del file che vogliamo che in output contenga il grafico creato dal comando **plot**.

## 11.7 Titolo del grafico

```
> set title '#1'
```

dove #1 è il titolo del grafico che apparirà sulla sua sommità.

## 11.8 Titoli degli assi

```
> set xlabel '#1'
> set ylabel '#2'
```

dove #1 è il titolo dell'asse x e #2 è il titolo dell'asse y.

# Parte V

# Esercitazioni

# Capitolo 2

# Rappresentazione dei numeri ed errori

## 2.1 Overflow, underflow e precisione

Determinare, sia tramite un proprio codice in Fortran (entro un fattore 10 e poi 2) che tramite gli appropriati comandi Fortran, i valori di Huge, Tiny e Epsilon per le rappresentazioni reali a 4B e 8B. Verificare il numero di cifre significative decimali tramite il comando Precision per le rappresentazioni reali a 4B e 8B.

Determinare sia tramite un proprio codice in Fortran (entro un fattore 10 e poi 2) che tramite gli appropriati comandi Fortran, il valore di Huge per le rappresentazioni intere a 4B e 8B. Verificare cosa accade superando con passo +1 (-1) il valore +Huge (-Huge) per le rappresentazioni intere a 4B e 8B.

## 2.2 Sommare una serie oscillante I

Determinare il valore di $e^{-x}$ per x = 0.1, 1, e 10 utilizzando la serie $e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!}$ con una precisione di $10^{-8}$. Utilizzare una serie finita $e^{-x} \cong \sum_{n=0}^{N} \frac{(-x)^n}{n!}$, determinando N in maniera tale che $\frac{\left|\frac{(-x)^N}{N!}\right|}{\sum_{n=0}^{N} \frac{(-x)^n}{n!}} < 10^{-8}$. Utilizzare i seguenti metodi ($e^{-x} \cong \sum_{n=0}^{N} a_n$):

1. $a_n = \frac{(-x)^n}{n!}$
2. $a_n = a_{n-1} \cdot \frac{-x}{n}$ con $a_0 = 1$ ed inizializzando la somma ad 1 ed iniziando a sommare da n = 1.
3. $e^{-x} = \frac{1}{e^x}$ e per $e^x$ usare la procedura 2. con x al posto di -x.
4. il comando Fortran exp(-x) [da considerarsi il valore vero con cui confrontarsi].

Determinare quale dei primi 3 metodi è quello più preciso e spiegare il perché.

## 2.3 Sommare una serie oscillante II

Determinare il valore della serie $A_N = \sum_{n=1}^{2N} (-1)^n \frac{n}{n+1}$ in funzione di N (tra 100,000 e 1,000,000 con passo 100,000 per 4B, da determinare in funzione dell'analisi per 8B) utilizzando: a) il metodo 1. dell'esercizio precedente, b) separando i contributi pari da quelli dispari, c) ricombinandoli in un'unica serie non oscillante. Graficare in log-log l'errore commesso calcolando con il metodo a), considerando il metodo c) come valore vero (i.e. $\log_{10}|(A_N^a - A_N^c)/A_N^c|$ vs. $\log_{10}N$).

## 2.4 Sommare una serie in su ed in giù

Determinare il valore della serie $A_N = \sum_{n=1}^{N} \frac{1}{n}$ in funzione di N (tra 100,000 e 1,000,000 con passo 100,000 per 4B, da determinare in funzione dell'analisi per 8B) sommando sia partendo da n = 1 per numeri crescenti sia partendo da n = N per numeri decrescenti. Effettuare il calcolo in entrambi le rappresentazioni reali a 4B e 8B. Le prime 6÷7 cifre del risultato nella rappresentazione a 8B può essere considerato il valore vero in rapporto al risultato nella rappresentazione a 4B. Determinare quale dei due modi di calcolo è il più preciso e spiegare il perché. Graficare in log-log l'errore commesso calcolando "in su", considerando il calcolo "in giù" come valore vero (i.e. $\log_{10}|(A_N^{su} - A_N^{giù})/A_N^{giù}|$ vs. $\log_{10}N$).

## 2.5 Cancellazione per sottrazione

Calcoliamo le soluzioni dell'equazione quadratica

$$ax^2 + bx + c = 0 \tag{2.1}$$

che sappiamo essere

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{o} \quad x_{1,2} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{2.2}$$

Se $a$ o $c$ sono molto piccoli rispetto a $b$ ($a \vee c \ll b$), la radice sarà praticamente pari a $|b|$ ed a seconda del segno di $b$, una delle due soluzioni avrà al numeratore o denominatore la sottrazione di due numeri praticamente identici con la conseguente perdita di precisione (vedi Sec. 2.5 a pagina 10).

Se utilizziamo invece le seguenti formule

$$q = -\frac{1}{2}\left(b + \text{sign}\,(b)\,\sqrt{b^2 - 4ac}\right) \qquad x_1 = \frac{c}{q} \quad x_2 = \frac{q}{a} \tag{2.3}$$

la sottrazione è scomparsa e la perdita di precisione scongiurata. E' possibile verificare il risultato confrontando le soluzioni numeriche ottenute nel limite di $a \vee c \ll b$ con quelle analitiche nello stesso limite

$$x_1 \approx -\frac{c}{b} \quad x_2 \approx -\frac{b}{a} + \frac{c}{b} \tag{2.4}$$

Utilizzare i valori $a = 1$, $b = 1$ e $c = 10^n$ con $n = -1, -2, -3, \ldots, -20$, sia in singola che in doppia precisione, e controllare la correttezza dei risultati provenienti dai quattro metodi di calcolo confrontandoli tra loro.

# Capitolo 3

# Interpolazione di funzioni

## 3.1 Interpolazione della funzione $\sin(x)$

Interpolare-estrapolare con il metodo di Aitken (e delle differenze, confrontando i due risultati) la funzione $\sin(x)$ tra $-2\pi$ e $4\pi$ assegnando 10, 20, 30, 40 e 50 punti equispaziati tra 0 e $2\pi$. Commentare i risultati.

## 3.2 Interpolazione della funzione $\log_{10}(x)$

Interpolare-estrapolare con il metodo di Aitken (e delle differenze, confrontando i due risultati) la funzione $\log_{10}(x)$ tra 0.001 e 20 assegnando 10, 20, 30, 40 e 50 punti equispaziati tra 0.1 e 10. Commentare i risultati.

## 3.3 Interpolazione della funzione $\frac{1}{x}$

Interpolare-estrapolare con il metodo di Aitken (e delle differenze, confrontando i due risultati) la funzione $\frac{1}{x}$ tra $-5$ e 5 assegnando 10, 20, 30, 40 e 50 punti equispaziati tra $-2$ e 2. Commentare i risultati.

# Capitolo 4

# Derivata numerica di funzioni

## 4.1 Derivare la funzione $\sin(x)$

Derivare la funzione $\sin(x)$ assegnando 10, 20, 30, 40 e 50 punti equispaziati tra 0 e $2\pi$ e verificare che la sua derivata prima coincide con $\cos(x)$ e che la sua derivata seconda coincide con $-\sin(x)$ in tutti i punti assegnati. Utilizzare l'estrapolazione ai bordi dell'intervallo dato sia con 10 che con tutti i punti dati. Confrontare i risultati ottenuti al variare del numero di punti utilizzati (10, 20, 30, 40 e 50), del numero di punti nelle formule (3 e 5), se il punto è interno o e ai bordi dell'intervallo, se lavoriamo in singola o doppia precisione.

## 4.2 Derivare la funzione $\log_{10}(x)$

Derivare la funzione $\log_{10}(x)$ assegnando 10, 20, 30, 40 e 50 punti equispaziati tra 0.1 e 10 e verificare che la sua derivata prima coincide con $\frac{1}{x}$ e che la sua derivata seconda coincide con $-\frac{1}{x^2}$ in tutti i punti assegnati. Utilizzare l'estrapolazione ai bordi dell'intervallo dato sia con 10 che con tutti i punti dati. Confrontare i risultati ottenuti al variare del numero di punti utilizzati (10, 20, 30, 40 e 50), del numero di punti nelle formule (3 e 5), se il punto è interno o e ai bordi dell'intervallo, se lavoriamo in singola o doppia precisione.

## 4.3 Derivare la funzione $\sin(x)$ con precisione data

Derivare la funzione $\sin(x)$ in 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$, $\frac{3\pi}{4}$, $\pi$ e verificare che la sua derivata prima coincide con $\cos(x)$ e che la sua derivata seconda coincide con $-\sin(x)$. Confrontare i risultati ottenuti al variare dello step iniziale $h$, della precisione "ricorsiva" richiesta ($10^{-6}$, $10^{-9}$, $10^{-12}$, $10^{-16}$), e se lavoriamo in singola o doppia precisione.

# Capitolo 5

# Integrazione numerica di funzioni

## 5.1 Integrare la funzione $\sin(x)$

Integrare la funzione $\sin(x)$ tra 0 e $\pi$ con $N$ dispari punti sia con la formula dei trapezi che con quella di Simpson e studiare l'errore relativo (usando il valore analitico esatto dell'integrale) in scala log-log per identificare il numero ottimale di punti $N$ utile per minimizzare l'errore (identificare anche il valore minimo dell'errore relativo $\varepsilon_{min}$) sia in singola che in doppia precisione.

## 5.2 Integrare la funzione $\sin(x)$ in maniera ricorsiva tramite i trapezi

Integrare la funzione $\sin(x)$ tra 0 e $\pi$ con la formula ricorsiva dei trapezi richiedendo un errore assoluto pari a $100\varepsilon_m$ sia in singola che in doppia precisione e confrontare il valore di punti utilizzati con quello ottimale ottenuto nell'esercitazione precedente e l'errore relativo raggiunto.

## 5.3 Integrare la funzione $\mathrm{e}^{-x}$ in maniera adattativa

Integrare la funzione $\mathrm{e}^{-x}$ tra $-10$ e 10 sia con la formula adattativa dei trapezi che con quella adattativa di Simpson richiedendo un errore assoluto pari a $10^{-4}$, $10^{-5}$ e $10^{-6}$ ed un numero massimo di punti pari 10 volte quello ottimale ottenuto nella prima esercitazione sia in singola che in doppia precisione e confrontare il valore ottenuto con quello analitico esatto.

## 5.4 Integrare la funzione $1/x$ in maniera adattativa

Integrare la funzione $1/x$ tra 1 e 10 sia con la formula adattativa dei trapezi che con quella adattativa di Simpson richiedendo un errore assoluto pari a $10^{-4}$, $10^{-5}$ e $10^{-6}$ ed un numero massimo di punti pari 10 volte quello ottimale ottenuto nella prima esercitazione sia in singola che in doppia precisione e confrontare il valore ottenuto con quello analitico esatto.

# Capitolo 6

# Ricerca degli zeri/minimo di una funzione

Trovare uno zero delle seguenti equazioni:

1. $\cos x = x$
2. $\log_{10} x = 0$
3. $\frac{1}{x} = x$
4. $\mathrm{e}^{-x} = x$
5. $\mathrm{e}^{x^2} \ln x^2 = x$
6. $\mathrm{e}^{x} \ln x = x^2$
7. Minimo di: $V(x) = a\left[\left(\frac{b}{x}\right)^{12} - \left(\frac{b}{x}\right)^{6}\right]$ per $a = b = 1$

# Elenco delle figure

# Elenco delle tabelle

# Bibliografia

[1] R.H. Landau, M.J. Páez, and C.C. Bordeianu, Computational Physics (Wiley-VCH, Weinheim, 2007) 1, 4, 5, 6, 7, 8, 10, 11

[2] T. Pang, An Introduction to Computational Physics (Cambridge University Press, Cambridge, 2006)

1, 13, 14, 15, 16, 17, 18, 19