Università degli studi Milano Bicocca - Dipartimento di Fisica

Esperimentazioni di Fisica Computazionale

S. Franceschina

June 8, 2025

Abstract

Contents

1 Analisi dell'errore

1.1 Teoria

Nella presente sezione analizziamo le due principali fonti di errore in contesti computazionali:

1. Errori di arrotondamento: dovuti alla rappresentazione di numeri reali con numero finito di digits.

2. Errori di approssimazione: dovuti alla modalità stessa con cui affrontiamo il problema, per questo motivo sono presenti anche nel caso ideale.

1.2 Esercizio 1.0.1

Considera la funzione $f(x) = e^x$ nell'intervallo $x \in [0, 1]$. Scrivi un programma che calcoli la serie approssimante:

$$g_N(x) = \sum_{n=0}^{N} \frac{x^n}{n!} \,. \tag{1}$$

1. Verifica che l'errore assoluto $\Delta = |f(x) - g_N(x)|$ scala approssimativamente come $x^{N+1}/(N+1)!$ per $N = 1, 2, 3, 4$.

2. L'errore Δ, nell'intervallo dato di x, differisce da $x^{N+1}/(N+1)!$. Perché accade questo e per quali valori di x?

1.2.1 Soluzione

Al fine dell'esercizio vengono rappresentati nel grafico 1 le funzioni Δ e $\frac{x^{N+1}}{(N+1)!}$, con $N = 1, 2, 3, 4$, al variare di x nell'intervallo $[0, 1]$.

La prima considerazione che possiamo fare è che all'aumentare di N la funzione Δ assume valori sempre più vicini allo zero. Questo significa che la distanza tra il valore della funzione $f(x)$ presa in esame e la sua espansione di Taylor troncata all'ordine N diminuisce. In effetti ci aspettiamo che la funzione Δ sia esattamente zero nel caso in cui $N \to \infty$. Inoltre possiamo notare che la funzione Δ, in ognuno dei grafici, è tanto più prossima allo zero quanto più ci si avvicina all'origine, poichè l'espansione in serie richiede $x \to 0$.
La seconda considerazione è che le funzioni Δ e $\frac{x^{N+1}}{(N+1)!}$ si avvicinano tra loro all'aumentare di N. Questo risponde alla richiesta dell'esercizio, cioè che l'errore scali come un polinomio di ordine $N + 1$.

1.3 Esercizio 1.2.1

Calcolare la seguente somma

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} = \lim_{N\to\infty} S(N) \quad \text{con} \quad S(N) = \sum_{n=1}^{N} \frac{1}{n^2} \tag{2}$$

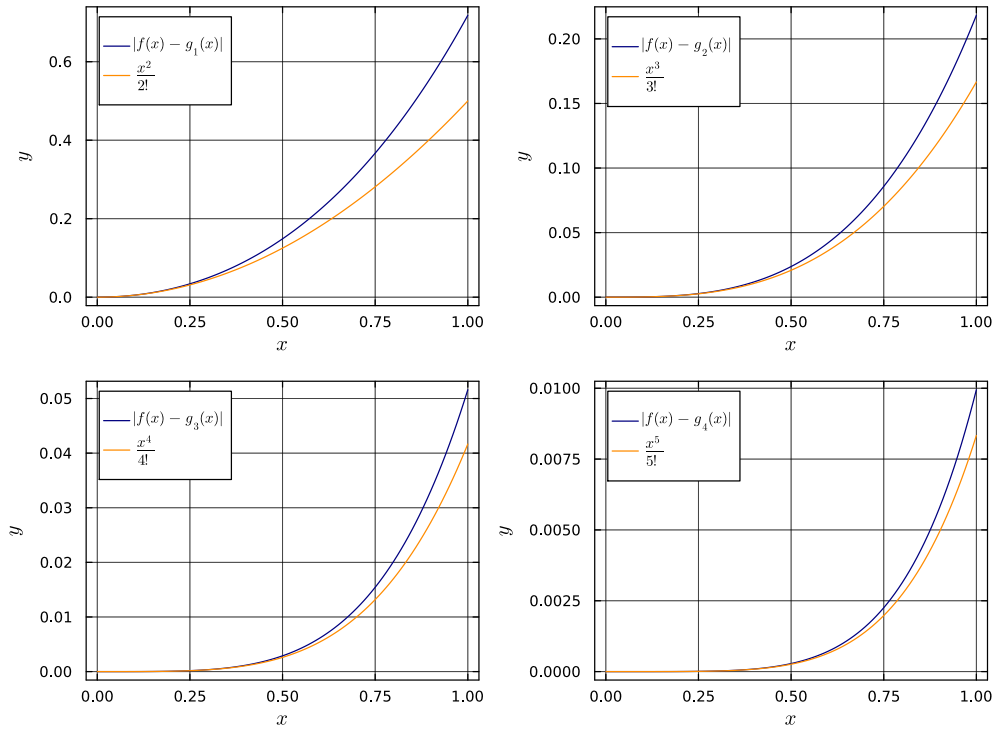

Figure 1: Confronto tra Δ e $\frac{x^{N+1}}{(N+1)!}$ per $N = 1, 2, 3, 4$.

1. Calcolare la somma in single precision utilizzando l'ordinamento normale, $n = 1, 2, 3, \ldots, N$.

2. Calcolare la somma in single precision utilizzando l'ordinamento inverso, $n = N, \ldots, 2, 1$.

3. Studiare la convergenza di entrambe le implementazioni in funzione di N tracciando il grafico di $|S(N) - \pi^2/6|$.

4. Ripetere i punti da 1 a 3 utilizzando double precision.

5. Sai spiegare cosa succede?

1.3.1 Soluzione

Procediamo prendendo in esame i due casi, single precision e double precision.

Single precision: Si riportano in figura 2 gli errori di troncamento della serie $\sum_{n=1}^{\infty} \frac{1}{n^2}$, laddove N è il troncamento. La somme sono state eseguite con variabili di tipo Float32 (single precision).

In figura 2 notiamo due comportamenti degni di nota. Il primo riguarda la curva blu, che arresta la sua discesa poco dopo $N = 4000$. Il secondo riguarda il discostarsi delle due curve, già a partire da $N \simeq 2000$, a causa dell'andamento a linea spezzata della curva blu.

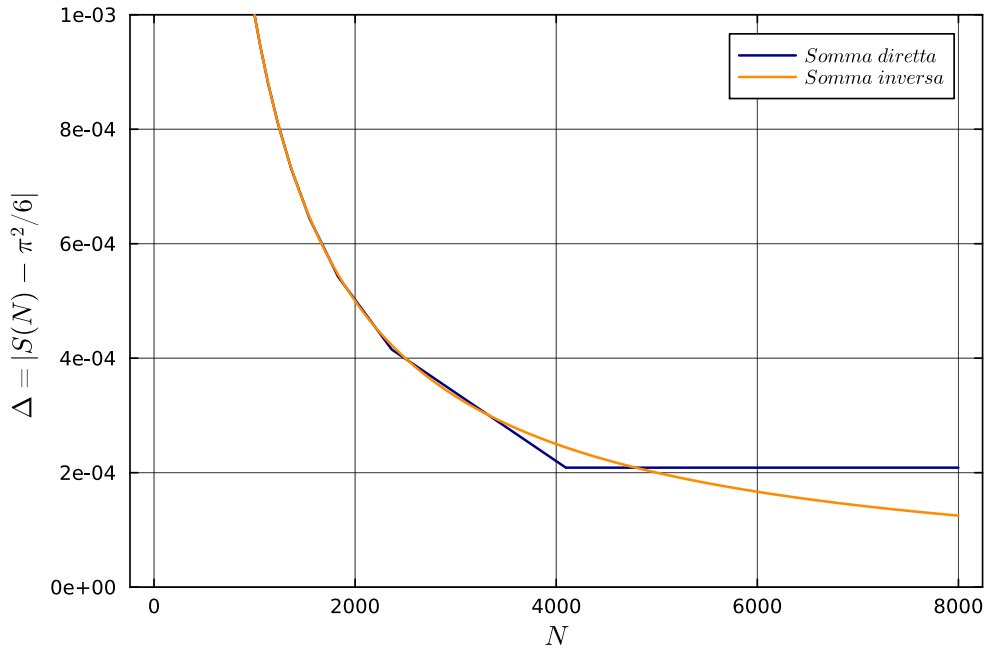

Figure 2: $S(N) = \sum_{n=1}^{N} \frac{1}{n^2}$ in single precision

La spiegazione del primo fenomeno è la seguente: nel caso della somma con ordinamento diretto, all'aumento dell'indice di somma, gli addendi sono sempre più piccoli. In particolare la somma comincia da 1, e sappiamo che dovrà raggiungere $\frac{\pi^2}{6} \simeq 1.64$ e perciò per tutto il processo la somma parziale resterà nell'intervallo $[1, 2)$. In tale intervallo la distanza tra due floating point numbers è $\epsilon_{mach} = 2^{-23}$. Osserviamo il grafico 2: la curva blu comincia ad essere costante a partire da $N = 4096 = 2^{12}$. Tale numero corrisponde all'addendo $\frac{1}{N^2} = 2^{-24}$, che è appena più piccolo di ϵ_{mach}, ovvero della distanza tra due floating point numbers in $[1, 2)$, e quindi è come sommare 0.
La spiegazione del secondo fenomeno è simile: in questo caso alla somma viene aggiunto un addendo che è più piccolo del precedente, ma non così tanto da essere più piccolo di ϵ_{mach}. In questo modo la somma viene migliorata, ma il nuovo addendo è arrotondato rispetto al suo valore vero. Il successivo addendo, pur essendo differente in teoria, a causa dell'arrotondamento risulta essere uguale al precedente. Ciò fa in modo che venga sommato sempre lo stesso numero, e così l'andamento della curva blu è lineare.
I due fenomeni non si verificano per la somma con ordinamento inverso perchè il primo numero ad essere sommato è molto piccolo. Questo fa in modo che i successivi floating point number siano molto vicini tra loro, è così sommare l'addendo successivo fa cadere la somma parziale vicina al floating point number che la approssima.

Si può mostrare matematicamente che gli errori $\Delta = |S(N) - \pi^2/6|$ scalano come $O(\frac{1}{N})$. Per verificare questa affermazione possiamo interpolare la curva degli errori con un metodo che vedremo nella sezione 2. Sulla base delle considerazioni di questa sezione, possiamo aspettarci che sia più conveniente interpolare la curva con ordinamento inverso, perchè i valori che la compongono sono meno affetti da errori di tipo numerico. Utilizzando il modello lin-

earizzato $log(\Delta) = -c * log(N)$ con $c = 1$ si ottiene per l'ordinamento inverso $c_{fit} = 1.00003$ e per quello diretto $c_{fit} = 0.986$. Come atteso il caso di ordinamento diretto restituisce un risultato peggiore di quello inverso.

Double precision: Si riporta in figura 3 un ingrandimento del grafico degli errori di troncamento della serie $\sum_{n=1}^{\infty} \frac{1}{n^2}$, laddove N è il troncamento. La somme sono state eseguite con variabili di tipo Float64 (double precision).

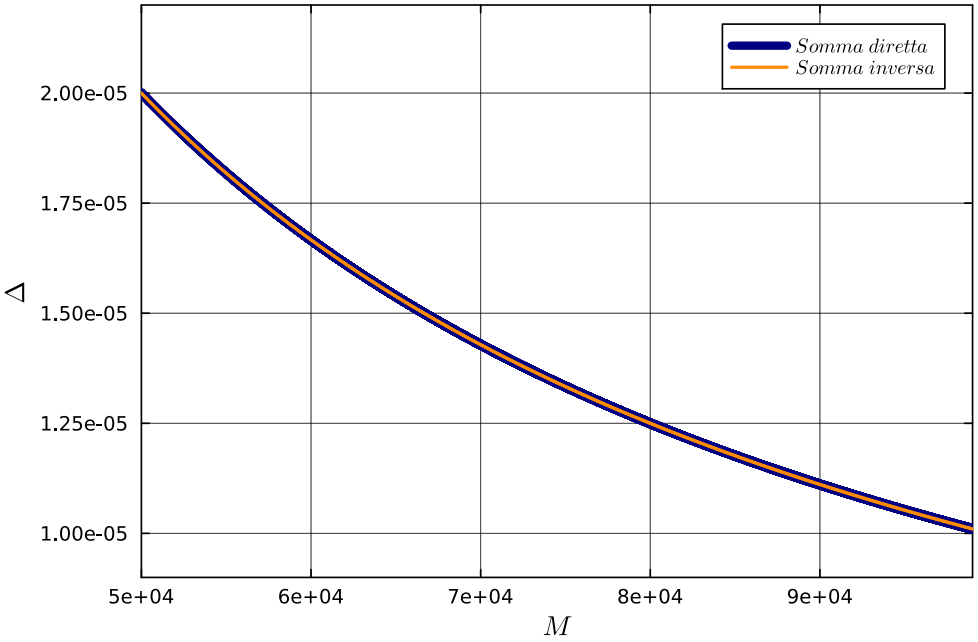

Figure 3: Ingrandimento di $S(N) = \sum_{n=1}^{N} \frac{1}{n^2}$ in double precision

Nel caso di figura 3 possiamo notare che le due curve sono sovrapposte, nonostante l'ingrandimento a valori di $N \in [5000, 10000]$. Per osservare un fenomeno simile a quello di figura 2 dovremmo raggiungere valori di $N = 2^{26}$. Infatti l'approssimazione non migliora quando vengono sommati numeri dell'ordine di ϵ_{mach} per double precision, cioè $\frac{1}{N^2} \simeq 2^{-52}$.

1.4 Esercizio 1.4.1

(a) In statistica, definiamo la varianza di un campione di valori $x_1, \ldots, x_n$ come

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2, \qquad \overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{3}$$

Scrivi una funzione che prenda in input un vettore x di lunghezza arbitraria e restituisca σ^2 calcolata con la formula sopra. Dovresti testare la funzione con $x = [1, 1, \ldots, 1]$ e con alcuni vettori casuali.

(b) Uno svantaggio della formula "naive" è che bisogna prima calcolare una somma per $\overline{x}$ prima di eseguire la somma per calcolare σ^2. Questo significa che bisogna scorrere due volte i dati. Ciò è indesiderabile per grandi insiemi di dati o quando la varianza campionaria deve essere calcolata mentre i dati vengono generati. Alcuni libri di testo riportano una formula a singolo ciclo:

$$\sigma^2 = \frac{1}{n-1}\left(u - \frac{1}{n}v^2\right), u = \sum_{i=1}^{n} x_i^2, \qquad v = \sum_{i=1}^{n} x_i. \tag{4}$$

Prova entrambe le formule per i seguenti dataset, ciascuno dei quali ha varianza esattamente uguale a 1. Esegui i calcoli sia in single che in double precision.
$x_1 = [1e3, 1 + 1e3, 2 + 1e3]$
$x_2 = [1e6, 1 + 1e6, 2 + 1e6]$
$x_3 = [1e7, 1 + 1e7, 2 + 1e7]$
$x_4 = [1e8, 1 + 1e8, 2 + 1e8]$
Sai spiegare i risultati?

1.4.1 Soluzione

(a) Si è testato il codice che implementa la formula 3 su tre vettori contenenti elementi di tipo Float64. Il primo contiene solo numeri 1, il secondo contiene numeri generati a partire da una distribuzione uniforme, il terzo numeri generati a partiere da una distribuzione normale. I valori sono riportati in tabella 1.

Table 1: Risultati del test della formula 3 su tre vettori di tipo Float64

Vettore	**Lunghezza**	**Varianza attesa**	**Varianza (output)**
Solo uno	10^3	0.000	0.000
Uniforme	10^8	0.083	0.083
Normale	10^8	1.000	1.000

I risultati di tabella 1 mostrano che l'implementazione del codice è corretta.

(b) Si è testato il codice che implementa la formula 4 su quattro vettori contenenti elementi di tipo Float32, Float64 e Long Double. I risultati sono riportati in tabella 2.

Table 2: Valori di varianza. Metodo 1: equazione 3. Metodo 2: equazione 4.

Set	σ^2 **attesa**	**Float32**		**Float64**		**Long Double**	
		Metodo 1	Metodo 2	Metodo 1	Metodo 2	Metodo 1	Metodo 2
x_1	1.0	1.0	1.0	1.0	1.0	1.0	1.0
x_2	1.0	1.0	-131072.0	1.0	1.0	1.0	1.0
x_3	1.0	1.0	0.0	1.0	1.0	1.0	1.0
x_4	1.0	0.0	0.0	1.0	0.0	1.0	1.0

I risultati di tabella 2 mostrano che il calcolo della varianza con la formula 4 è più affetto dalla mancanza di precisione della rappresentazione dei numeri durante il calcolo. Infatti,

mano a mano che la la precisione dei numeri aumenta, il risultato della formula 4 si avvicina a quello atteso.
C'è da notare che anche la formula 3 è affetta dalla stessa problematica, ma in maniera meno evidente. Basti osservare il risultato del metodo 1 per il vettore x_4 in Float32, che è 0.0, contro l'atteso 1.0.
Parte del problema risiede nella cancellazione numerica. Infatti, per valori degli elementi del vettore molto grandi, la differenza tra il quadrato della sommma e la somma dei quadrati è molto piccola e incorre in problema di cancellazione. Per verificare questa affermazione, si riportano in tabella 3 i valori della differenza $u - \frac{v^2}{n}$ per la formula 4.

Table 3: Differenza $u - \frac{v^2}{n}$ per ciascun dataset.

Dataset	**Single**	**Double**	**Long Double**
x_1	2.0	2.0	2.0
x_2	-262144.0	2.0	2.0
x_3	0.0	2.0	2.0
x_4	0.0	0.0	2.0

Leggendo la tabella 3 possiamo notare che per il dataset x_1 la differenza è esattamente 2.0, che è il risultato atteso. Per il dataset x_2 la differenza è molto grande, -262144.0, il che indica un significativo problema di cancellazione numerica. Per il dataset x_3 e x_4, la differenza è 0.0, indice del fatto che i singoli valori $u - \frac{v^2}{n}$ sono vicini tra loro meno di $\epsilon_m ach$, e perciò la loro differenza è zero.
Per concludere, possiamo dire che la formula 4 è più efficiente in termini di tempo di calcolo, ma è più affetta da errori numerici rispetto alla formula 3.

1.5 Esercizio 1.4.2

Sia $f(x) = \frac{e^x - 1}{x}$.

(a) Trova il numero di condizionamento $\kappa_f(x)$. Qual è il massimo di $\kappa_f(x)$ nell'intervallo $-1 \leq x \leq 1$?

(b) Usa l'algoritmo "naive"

$$f(x) = \frac{e^x - 1}{x} \tag{5}$$

per calcolare $f(x)$ per $x = 10^{-3}, 10^{-4}, 10^{-5}, \ldots, 10^{-16}$.

(c) Crea un secondo algoritmo utilizzando i primi n termini della serie di Maclaurin, cioè

$$p(x) = 1 + \frac{1}{2!}x + \frac{1}{3!}x^2 + \cdots + \frac{1}{(n+1)!}x^n. \tag{6}$$

Valutalo sugli stessi valori di x del punto (b). Per farlo devi scegliere un valore per n. Verifica la stabilità del risultato al variare di n. Avresti potuto indovinare un buon valore di n fin dall'inizio?

(d) Confronta i risultati delle due implementazioni in funzione di x. Quale algoritmo pensi sia più accurato, e perché?

1.5.1 Soluzione

(a) Il numero di condizionamento di una funzione f è definito come:

$$\kappa_f(x) = \left| \frac{xf'(x)}{f(x)} \right| . \tag{7}$$

Per la funzione $f(x) = \frac{e^x - 1}{x}$, calcoliamo la derivata:

$$f'(x) = \frac{e^x x - (e^x - 1)}{x^2} = \frac{e^x(x-1)+1}{x^2} . \tag{8}$$

Quindi il numero di condizionamento diventa:

$$\kappa_f(x) = \left| \frac{x\left(\frac{e^x(x-1)+1}{x^2}\right)}{\frac{e^x-1}{x}} \right| = \left| \frac{e^x(x-1)+1}{(e^x-1)x} \right| = \left| \frac{e^x x}{(e^x-1)} - 1 \right| . \tag{9}$$

La derivata di $k_f(x)$ non si annulla mai nell'intervallo $-1 \leq x \leq 1$, e $\kappa_f(0) = 0$. Dato che $\kappa_f(x)$ è monotona crescente per $x \geq 0$ e monotona decrescente per $x < 0$, il massimo si ha in $x = 1$, dove assume il valore $\kappa_f(1) = 0.58198$.

(b)-(c): Si riportano in figura 4 i risultati dei due algoritmi. La dicitura $f(x)$ si riferisce alla funzione calcolata con l'algoritmo "naive", mentre $p_n(x)$ si riferisce alla funzione calcolata con la serie di Maclaurin, fino al termine n.

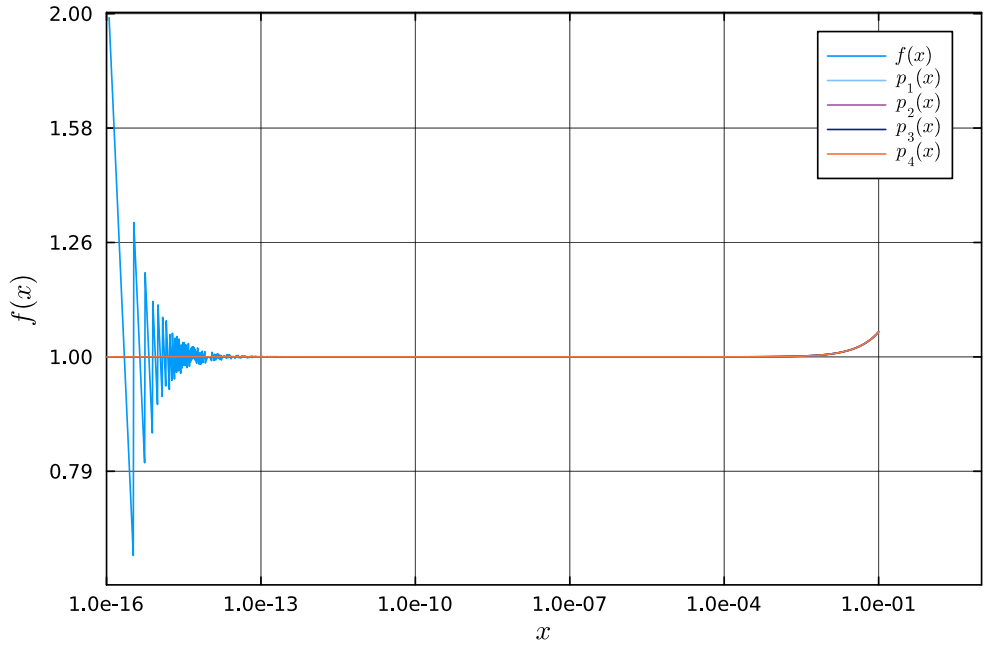

Figure 4: Confronto tra $f(x)$ e $p_n(x)$ al variare di x.

Per uno studio più accurato, si riporta in figura 5 un ingrandimento per piccoli valori di x (range $[10^{-15}, 10^{-14}]$) della figura 4 e in figura 6 un ingrandimento per valori di x (range $[10^{-2}, 10^{-1}]$) della figura 4.

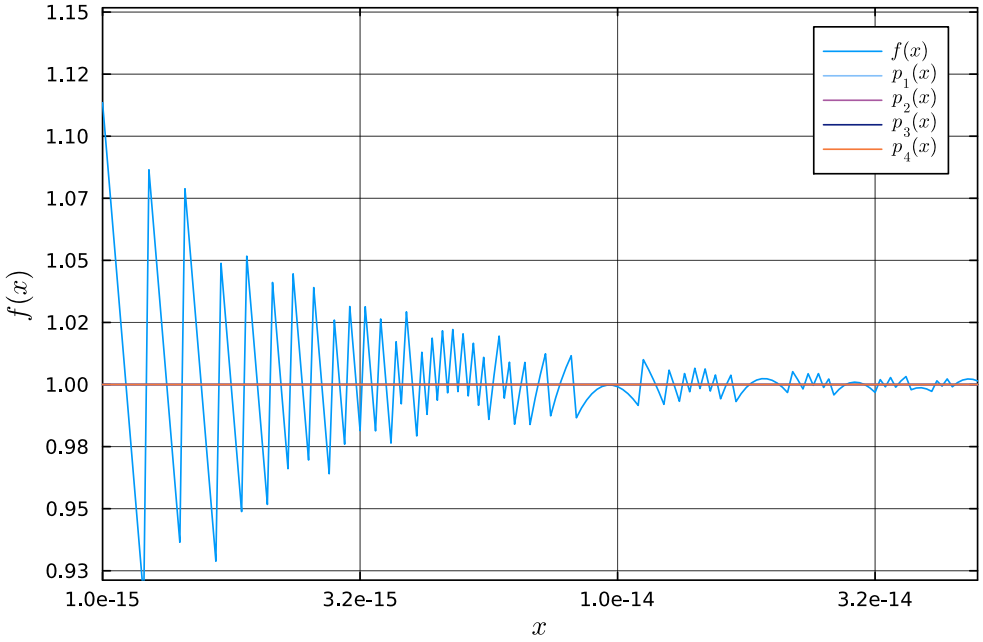

Figure 5: Ingrandimento di $f(x)$ e $p_n(x)$ per piccoli valori di x.

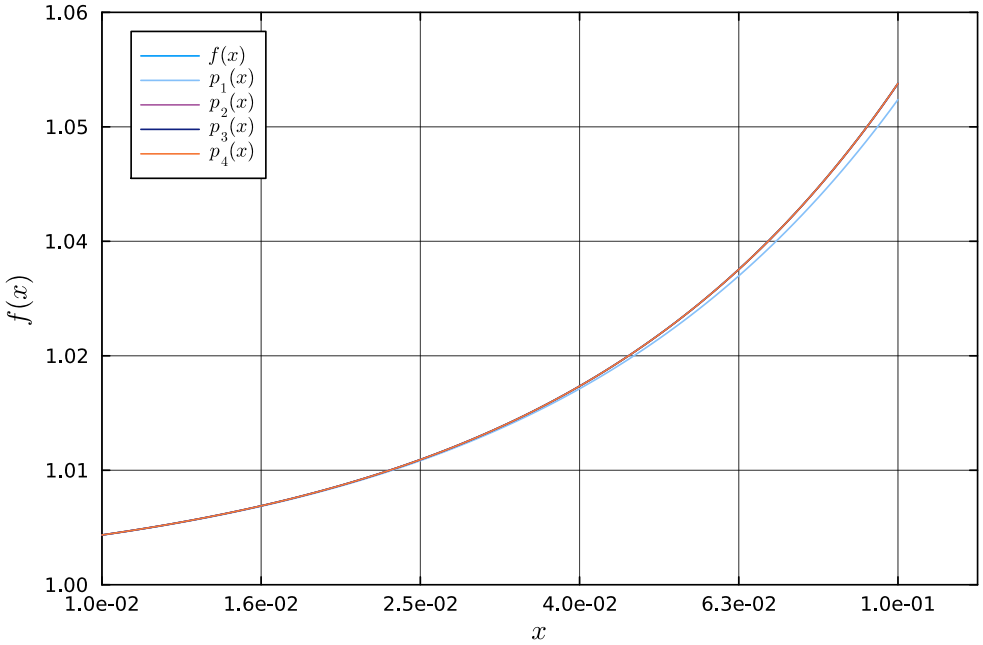

Figure 6: Ingrandimento di $f(x)$ e $p_n(x)$ per valori di x nell'intervallo $[10^{-2}, 10^{-1}]$.

Per quanto riguarda il comportamento della funzione $f(x)$ calcolata con l'algoritmo 5, possiamo notare che per valori di x piccoli, in particolare nella regione rappresentata in figura 5, la funzione assume valori non corretti, nonostante $\kappa_f(x)$ abbia un massimo non molto grande e peraltro non assunto nell'intorno di zero. La spiegazione di questo fenomeno

è che l'algoritmo 5, nonostante non abbia $\kappa_f(x)$ complessivo molto grande, è affetto da errori di cancellazione numerica a causa della forma in cui è scritta la funzione, ma non a causa del fatto che il problema sia intrinsecamente instabile. Di fatto possiamo intuire che esista una possibile riscrittura della funzione che non sia affetta da cancellazione numerica, e che sia più stabile, proprio perchè sappiamo che il massimo di $\kappa_f(x)$ non è molto grande. Tale riscrittura è quella che utilizza la serie di Maclaurin, implementata nell'algoritmo $p_n(x)$.
Possiamo notare che l'algoritmo 6, per valori di x piccoli, restituisce il valore corretto, dato che non è affetto da cancellazione numerica. Al contrario, nella regione ingradita di figura 6, l'algoritmo 6 comincia a restituire valori errati perchè ci allontaniamo dalla regione di convergenza della serie di Maclaurin.
Osservando la figura 6, notiamo che, per valori di x fissati, l'algoritmo 6 è stabile, cioè le curve che descrivono i valori di $p_n(x)$ sono sempre più vicine tra loro all'aumentare di n, fino a sovrapporsi. Per stimare un buon valore di n da utilizzare per calcolare $p_n(x)$, possiamo osservare che la serie di Maclaurin è tanto meno stabile quanto più ci si allontana da zero. Nel caso di studio valutiamo la distanza tra i valori di $p_n(x)$ al variare di n, fissato $x = 0.1$. Si è rappresentato in figura 7 l'andamento della distanza tra i valori di $p_n(0.1)$ e $p_{n+1}(0.1)$ al variare di n. In formula:

$$\Delta_n = |p_n(0.1) - p_{n+1}(0.1)| \,. \tag{10}$$

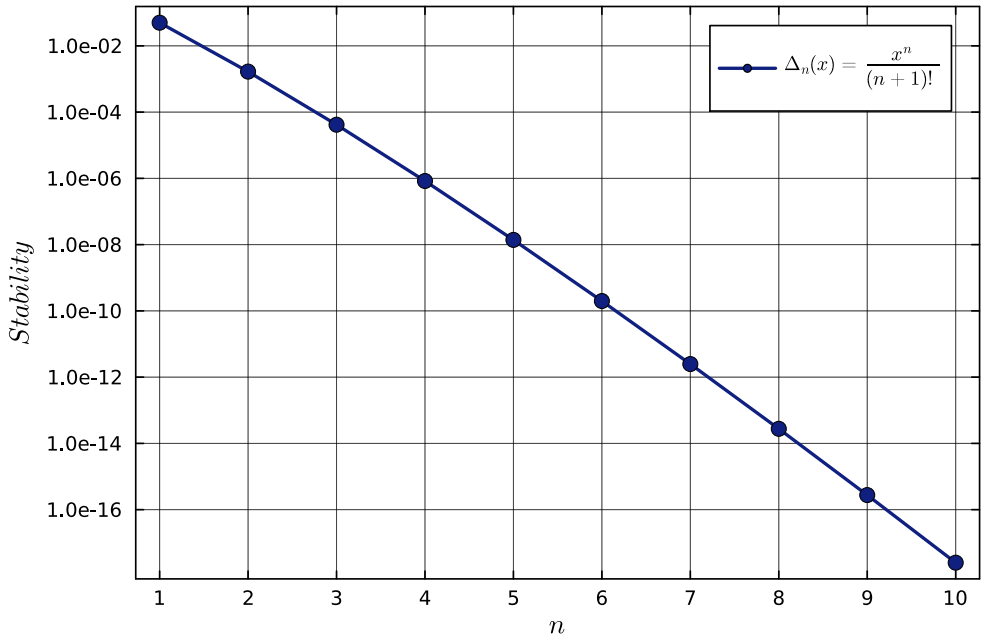

Figure 7: Distanza tra $p_n(0.1)$ e $p_{n+1}(0.1)$ al variare di n.

Ai fini di questo esercizio possiamo dire che $n = 2$ è già un buon valore perchè le distanze tra i polinomi successivi non sono apprezzabili nel range $[1, 1.06]$, caratteristico della figura 6. È chiaro che volendo raggiungere precisione massima occorrerà scegliere $n = 10$, dato che Δ_n corrispondente è più piccolo di $\epsilon mach$

(d): Come confronto tra i due algoritmi si riporta in figura 7 il grafico dell'errore tra i due algoritmi, cioè

$$\Delta(x) = |f(x) - p_n(x)| \ . \tag{11}$$

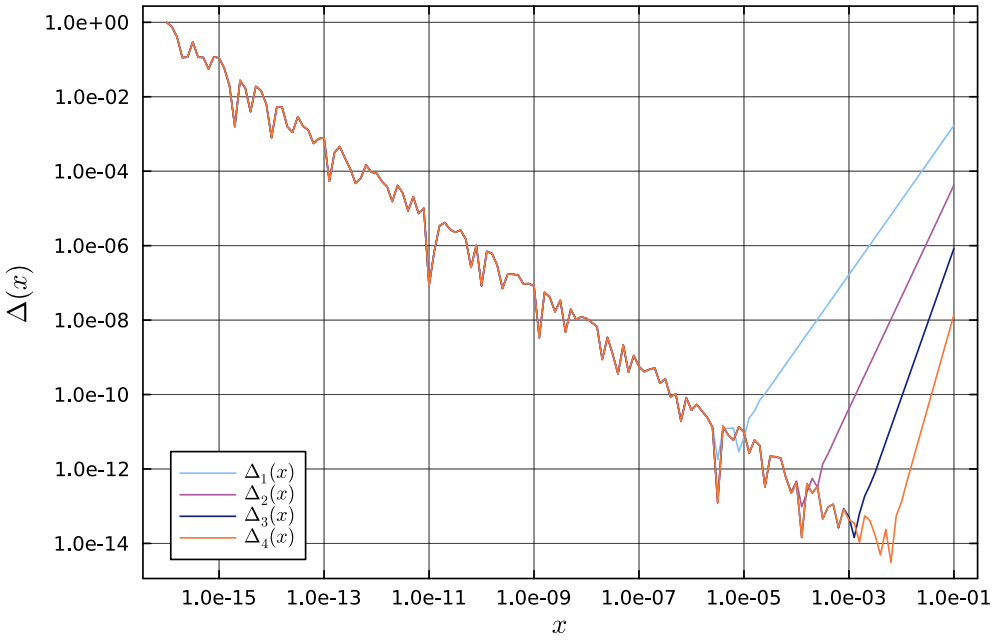

Figure 8: Distanza tra $f(x)$ e $p_n(x)$.

La figura 8 mostra una evidente differenza tra i due algoritmi nella regione di instabilità numerica di dell'algoritmo 5, cioè per valori di x piccoli. E' interessante notare che la distanza tra i due algoritmi diminuisce all'aumentare di x, ma torna ad aumentare a partire da $x = 10^{-5}$, laddove l'algoritmo 6 esce dalla regione di convergenza della serie di Maclaurin. In conclusione, possiamo affermare che l'algoritmo 6 è più accurato per valori di x piccoli, mentre l'algoritmo 5 è più accurato per valori di x grandi.

2 Sistemi lineari

2.1 Teoria

2.2 Esercizi 2.1.1, 2.1.2, 2.1.3, 2.1.4

Scrivi una funzione che esegua la sostituzione in avanti su una matrice triangolare inferiore $n \times n$.

2. Testa il tuo codice sui seguenti sistemi lineari:

(a) $\begin{bmatrix} -2 & 0 & 0 \\ 1 & -1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -4 \\ 2 \\ 1 \end{bmatrix}$

(b) $\begin{bmatrix} 4 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ -1 & 4 & 4 & 0 \\ 2 & -5 & 5 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -4 \\ 1 \\ -3 \\ 5 \end{bmatrix}$

Puoi verificare la soluzione risolvendo il sistema a mano e/o calcolando $\mathbf{Lx}$ e sottraendo $\mathbf{b}$.

3. Scrivi una funzione che esegua la sostituzione all'indietro su una matrice triangolare superiore $n \times n$.

4. Testa il tuo codice sui seguenti sistemi lineari:

(a) $\begin{bmatrix} 3 & 1 & 0 \\ 0 & -1 & -2 \\ 0 & 0 & 3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 6 \end{bmatrix}$

(b) $\begin{bmatrix} 3 & 1 & 0 & 6 \\ 0 & -1 & -2 & 7 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 5 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 4 \\ 1 \\ 1 \\ 5 \end{bmatrix}$

Puoi verificare la soluzione risolvendo il sistema a mano e/o calcolando $\mathbf{Ux}$ e sottraendo $\mathbf{b}$.

2.2.1 Soluzione

Per risolvere i sistemi lineari proposti, si sono implementate le funzioni di sostituzione in avanti e all'indietro. Di seguito sono riportati i risultati delle soluzioni.

3 Radici di equazioni non lineari

3.1 Teoria

4 Interpolazioni

4.1 Teoria

5 Integrazione numerica

5.1 Teoria

6 Equazioni differenziali ordinarie

6.1 Teoria