

1 Lezione del 03-03-25

1.1 Riassunto sulla stima dell'errore

Riassumiamo quindi le regole viste per la stima dell'errore su funzioni razionali. Avevamo dato la definizione di errore **assoluto** σ_f e errore **relativo** ϵ_f , entrambi composti da due fattori denominati errore **algoritmico** e errore **inerente**, con pedici rispettivamente a e d .

- Riguardo all'errore **inerente assoluto** avevamo preso su un dominio D la stime:

$$|\sigma_d| \leq \sum_{j=1}^m A_j \cdot |\sigma_j|$$

con $|\sigma_j|$ **errore di arrotondamento** e A_j **coefficiente di amplificazione**:

$$A_j = \max_{P \in D} \left(\frac{\partial f}{\partial x_j}(P) \right)$$

Per l'errore di arrotondamento avevamo visto potevamo prendere:

$$|\sigma_j| \leq U \cdot |x_j|$$

con U precisione macchina.

- Riguardo all'**errore inerente relativo** avevamo invece preso:

$$|\epsilon_d| \leq \sum_{j=1}^m \overline{A}_j \cdot |\epsilon_j|$$

con $|\epsilon_j|$ **errore di arrotondamento relativo** e \overline{A}_j **coefficiente di amplificazione relativo**:

$$\overline{A}_j = \max_{P \in D} \left(\frac{x_j \cdot \frac{\partial f}{\partial x_j}(P)}{f(P)} \right)$$

Per l'errore di arrotondamento relativo potevamo quindi prendere:

$$|\epsilon_j| \leq U$$

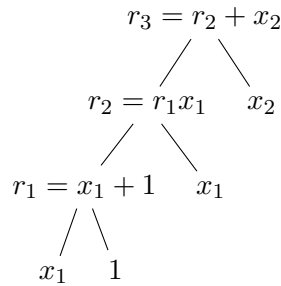
Vediamo come ultimo esempio il calcolo dell'errore relativo $|\epsilon_f|$ per la funzione:

$$f(x_1, x_2) = (x_1 + 1)x_1 + x_2$$

- Iniziamo con il calcolo dell'errore inerente, senza fare assunzioni su ϵ_{x_i} :

$$|\epsilon_d| \leq \frac{x_1 \cdot \frac{\partial f}{\partial x_1}(x_1, x_2)}{f(x_1, x_2)} \epsilon_{x_1} + \frac{x_2 \cdot \frac{\partial f}{\partial x_2}(x_1, x_2)}{f(x_1, x_2)} \epsilon_{x_2} = \frac{x_1(2x_1 + 1)}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_2}$$

- Calcoliamo poi l'errore algoritmico dell'algoritmo:



da cui:

$$\begin{aligned}
 |\epsilon_a| &\leq \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} \epsilon_{r_2} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_2} = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_{r_1} + \epsilon_{x_1}) \\
 &= \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1 + \frac{x_1}{x_1 + 1} \epsilon_{x_1} + \frac{1}{x_1 + 1} \cdot 0) = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1)
 \end{aligned}$$

Abbiamo quindi l'errore complessivo:

$$|\epsilon_f| \leq |\epsilon_a| + |\epsilon_d| = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1) + \frac{x_1(2x_1 + 1)}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1}$$

1.2 Errori di funzioni non razionali

Abbiamo finora trascurato il caso di funzioni non razionali. Prendiamo ad esempio di voler calcolare l'errore su funzioni come $e^{\cos(x+y)}$. In questo caso sarà necessario usare un'approssimazione razionale di f che chiamiamo \bar{f} , che poi porteremo a \bar{f}_a che usa operazioni macchina detta **algoritmo**. In questo caso l'errore sarà dato dall'*errore inerente*, dall'*errore algoritmico* e dall'**errore analitico** σ_{an} della funzione, cioè potremo dire:

$$\bar{f}_a(P_1) - f(P_0) = \bar{f}_a(P_1) - \bar{f}(P_1) + \bar{f}(P_1) - f(P_1) + f(P_1) - f(P_0) = \sigma_a + \sigma_{an} + \sigma_d$$

L'errore inerente sarà calcolato sulla f originale, mentre l'errore analitico sarà calcolato con la nuova \bar{f} , e in particolare dipenderà dall'approssimazione razionale che usiamo.

Vediamo per adesso approssimazioni polinomiali attraverso la **formula di Taylor**. Nel caso scalare si ha:

Teorema 1.1: Formula di Taylor

Data $f : \mathbb{R} \rightarrow \mathbb{R}$, $f \in C^1$, allora dato $x_0 \in \mathbb{R}$ si ha:

$$f(x) = \sum_{n=0}^k \frac{f^{(n)}(x_0)}{n!} \cdot (x - x_0)^n + \frac{f^{(k+1)}(\eta)}{(k+1)!} (x - x_0)^{k+1}$$

Dove:

$$\epsilon_l = \frac{f^{(k+1)}(\eta)}{(k+1)!} (x - x_0)^{k+1}$$

rappresenta l'**errore di Lagrange** al k -esimo grado, con $\eta \in [x_0, x]$ il punto di massimo della $k+1$ -esima derivata di f .

In questo caso:

$$\bar{f}(x) = \sum_{n=0}^k \frac{f^{(n)}(x_0)}{n!} \cdot (x - x_0)^n$$

cioè la serie di Taylor troncata al k -esimo grado sarà una buona approssimazione per f , e l'errore analitico sarà dato da:

$$\sigma_{an} = R(x) = f(x) - T(x, k, x_0)$$

con $R(x)$ il resto fra lo sviluppo di Taylor troncato $T(x, k)$ e la funzione stessa $f(x)$. In questo caso fissato k si potrà dare una stima di errore direttamente dall'errore di Lagrange, cioè:

$$R(x) = f(x) - T(x, k, x_0) = \frac{f^{(k+1)}(\eta)}{(k+1)!} (x - x_0)^{k+1}$$

e più in particolare, sfruttando il limite superiore dato da:

$$R(x) \leq \epsilon_l = \frac{\max_{[x_0, x]} (f^{(k+1)}(\eta))}{(k+1)!} (x - x_0)^{k+1} = \frac{M}{(k+1)!} (x - x_0)^{k+1}$$

1.2.1 Approssimazione dell'esponenziale

Prendiamo di volere calcolare l'errore dato dall'approssimazione dell'esponenziale al k -esimo grado su un intervallo comprensivo di $x_0 = 0$, che chiamiamo $D = [-l, u]$. Avremo quindi l'approssimazione:

$$e^x = T(x, k, 0) = \sum_{n=0}^k \frac{(x - x_0)^n}{n!}$$

e la funzione resto:

$$R(x) = e^x - T(x, k, 0) = \frac{f^{(k+1)}(\eta)}{(k+1)!} x^{k+1}$$

prendiamo quindi la stima superiore di $f^{(k+1)}(\eta)$ su $\eta \in D$ (il caso peggiore sarà quello in cui valutiamo x sull'estremo superiore u):

$$f^{(k+1)}(\eta) \leq \max_{\eta \in [0, u]} f^{(k+1)}(\eta) = e^u$$

da cui:

$$R(x) \leq \frac{e^u x^{k+1}}{(k+1)!}$$

Stimiamo allora l'errore analitico relativo come:

$$|\epsilon_{an}| = \frac{R(x)}{f(x)} \leq \frac{\max_{[-l, u]} \left| \frac{e^u x^{k+1}}{(k+1)!} \right|}{\min_{[-l, u]} e^x} = \frac{e^{u-l} u^{k+1}}{(k+1)!}$$

1.2.2 Note di implementazione

Vediamo alcuni dettagli sull'implementazione pratica di un'approssimazione dell'esponenziale nel linguaggio MATLAB/Octave. Il primo modo che potrebbe venire in mente prevede di mantenere un numeratore e un denominatore:

```

1 function val = myexp1(x, n)
2     num = 1;
3     den = 1;
4
5     val = 1;
6
7     for i = 1:n
8         num = num * x;
9         den = den * i;
10
11         val = val + num / den;
12     end
13 end

```

Notiamo però che questo approccio presenta notevole instabilità numerica con grandi valori di k , ad esempio si veda:

```

1 >> myexp(20, 500)
2 ans =
3     NaN

```

Questo deriva dal fatto che per grandi valori di k si ottiene eventualmente $\text{num} = \text{Inf}$ e $\text{den} = \text{Inf}$, da cui il **NaN**. Un'approccio migliore si ha quindi mantenendo direttamente il termine e accumulandolo:

```

1 function acc = myexp2(x, n)
2     term = 1;
3     acc = 1;
4
5     for i = 1:n
6         term = term * x / i;
7         acc = acc + term;
8     end
9 end

```

Notiamo però che in questo caso si hanno problemi di cancellazione per argomenti negativi, ad esempio si veda:

```

1 >> myexp(-30, 500)
2 ans =
3     -3.0668e-05

```

In questo caso si ha più accuratezza imponendo argomenti positivi, ad esempio sfruttando:

$$e^{-x} = \frac{1}{e^x}$$

Si veda ad esempio:

```

1 >> 1/myexp(30, 500)
2 ans =
3     9.3576e-14

```

Modifichiamo quindi la funzione per rilevare automaticamente esponenti negativi e adottare la forma corretta:

```

1 function acc = myexp3(x, n)
2     neg = false;
3     if x < 0
4         neg = true;
5         x = -x;
6     end
7
8     term = 1;
9     acc = 1;
10
11    for i = 1:n
12        term = term * x / i;
13        acc = acc + term;
14    end
15
16    if neg
17        acc = 1 / acc;
18    end
19 end

```

Provando questa funzione su un range di interi da -30 a 30 , con la serie di Taylor troncata a 500, dà un errore massimo rispetto al valore reale di circa $10^{-16} \sim 10^{-17}$.

1.3 Richiami di algebra lineare

Nella maggior parte dei casi che ci interessano vorremo trattare non di scalari, ma di quantità vettoriali, ad esempio $x \in \mathbb{C}^n$, con $n > 1$.

1.3.1 Matrici complesse

Ci saranno utili le matrici perchè rappresentano direttamente tutte le **funzioni lineari**. Ad esempio, posta $f : \mathbb{C}^n \rightarrow \mathbb{C}^m$ tale che:

- $f(x + y) = f(x) + f(y)$ (addittività);
- $f(\lambda x) = \lambda f(x)$ (omogeneità)

detta *funzione lineare* allora $\exists! A \in \mathbb{C}^{m \times n}$ tale che $f(x) = Ax, \forall x \in \mathbb{C}^n$.

Nel corso useremo sia matrici in \mathbb{R} che matrici in \mathbb{C} , dove l'appartenenza a ciascuno di questi campi dipende dalla appartenenza di essi delle **entrate** A_{ij} della matrice. In ogni caso, una matrice reale non sarà che un caso particolare delle matrici complesse.

Si danno poi per scontate le definizioni di matrici:

- *Quadrate* ($n = m$);
- *Rettangolari* ($n \neq m$);
- *Diagonali* (elementi nulli fuori dalla diagonale);
- *Triangolari superiori/inferiori* (elementi nulli sotto/sopra la diagonale)

1.3.2 Indipendenza lineare

Diamo la definizione di indipendenza lineare:

Definizione 1.1:

Un insieme di vettori $\{x_1, \dots, x_s\}$ si dice linearmente indipendente se:

$$x_1 + \dots + x_s = 0 \leftrightarrow x_1, \dots, x_s = 0$$

Inoltre, se $s = n$ l'insieme $\{x_1, \dots, x_s\}$ si dice **base** di \mathbb{C}^n , e cioè $\forall y \in \mathbb{C}^n \exists! \{c_1, \dots, c_s\}$ tali che $y = \sum_{j=1}^n c_j x_j$.

1.3.3 Prodotto scalare

Diamo la definizione di prodotto scalare, generalizzato al campo complesso dal **prodotto hermitiano** (entrambi *prodotti interni*):

Definizione 1.2: Prodotto interno

Definiamo il prodotto interno fra due vettori $x, y \in \mathbb{C}^n$ come:

$$\langle x, y \rangle = \sum_{j=1}^n x_j \overline{y_j}$$

dove $\overline{y_j}$ rappresenta il **coniugato** di y_j , che chiaramente in \mathbb{R} si riduce a y_j stesso e quindi:

$$\langle x, y \rangle = \sum_{j=1}^n x_j y_j$$

1.4 Trasposta coniugata

Definiamo infine la **trasposta coniugata** di una certa matrice, generalizzata al campo complesso dalla **matrice hermitiana**:

Definizione 1.3: Trasposta coniugata

Data una matrice $A \in \mathbb{C}^{n \times m}$, la trasposta coniugata A^T sarà:

$$(A^T)_{ij} = A_{ji}$$

e la matrice hermitiana A^H sarà:

$$(A^H)_{ij} = \overline{A_{ji}}$$

1.4.1 Operazioni matriciali

Date due matrici A e B con lo stesso numero di righe e colonne si possono definire le operazioni:

- **Somma** ($A, B, C \in \mathbb{C}^{m \times n}$, $A + B = C$, $c_{ij} = a_{ij} + b_{ij}$);
- **Prodotto** ($A \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{n \times p}$, $C \in \mathbb{C}^{m \times p}$, $A \cdot B = C$, $c_{ij} = \sum_{h=1}^n a_{ih} b_{hj}$ sia in reali che in complessi).

Dal punto di vista computazionale, si ha che il prodotto scalare ha complessità $O(n)$, e la moltiplicazione matriciale ha complessità $O(m, n)$.