Esercitazione 2 11/03/2020

Esempio: $f(x) = x^n = e^{n \log(x)}$ $(x > 0)$

(1) Condizionamento del problema

(2) Stabilità degli algoritmi

(3) Costo computazionale. (stimiamo il numero di operazioni)

Condizionamento $f(x) = x^n$

$$\epsilon_m \doteq \frac{f'(x)}{f(x)} x \cdot \epsilon_x \qquad |\epsilon_x| \leq u$$

$$\epsilon_m \doteq \frac{n x^{n-1} \cdot x}{x^n} \epsilon_x = n \epsilon_x$$

$$|\epsilon_m| \doteq |n \epsilon_x| \leq n \cdot u$$

$$C_x = n$$

IL PROBLEMA È BEN CONDIZIONATO

($|\epsilon_m| \leq p(n) u$ p polinomio)

ERRORE ALGORITMICO

$$f(x) = x^n = (((x \cdot x)x) \cdot x \dots \cdot)x$$

```
p = 1;
for k = 1 : n
    p = p * x
end.
```

$x \cdot x \rightsquigarrow x \otimes x = x^2 \cdot (1+\epsilon_1) \quad |\epsilon_1| \le u$ (errore locale nella singola moltiplicazione)

$(x \in F(B,t,m,M)$ errore dipende solo dalla precisione di macchina)

$x \cdot x \cdot x \rightsquigarrow (x \otimes x) \otimes x = x^3 \cdot (1+\epsilon_1)(1+\epsilon_2)$ (errore totale)

$\vdots$

$\underbrace{x \dots \cdot x}_{n \text{ volte}} \rightsquigarrow x^n \cdot \prod_{i=1}^{n-1} (1+\epsilon_i) \quad |\epsilon_i| \le u$

$$g(x) = x^n \cdot \prod_{i=1}^{n-1} (1+\epsilon_i) \doteq x^n \cdot \left(1 + \sum_{i=1}^{n-1} \epsilon_i\right)$$

$$\epsilon_{alg} = \frac{g(x) - f(x)}{f(x)}$$

[Nella definizione $\tilde{x} = x$ perché stiamo studiando l'errore introdotto]

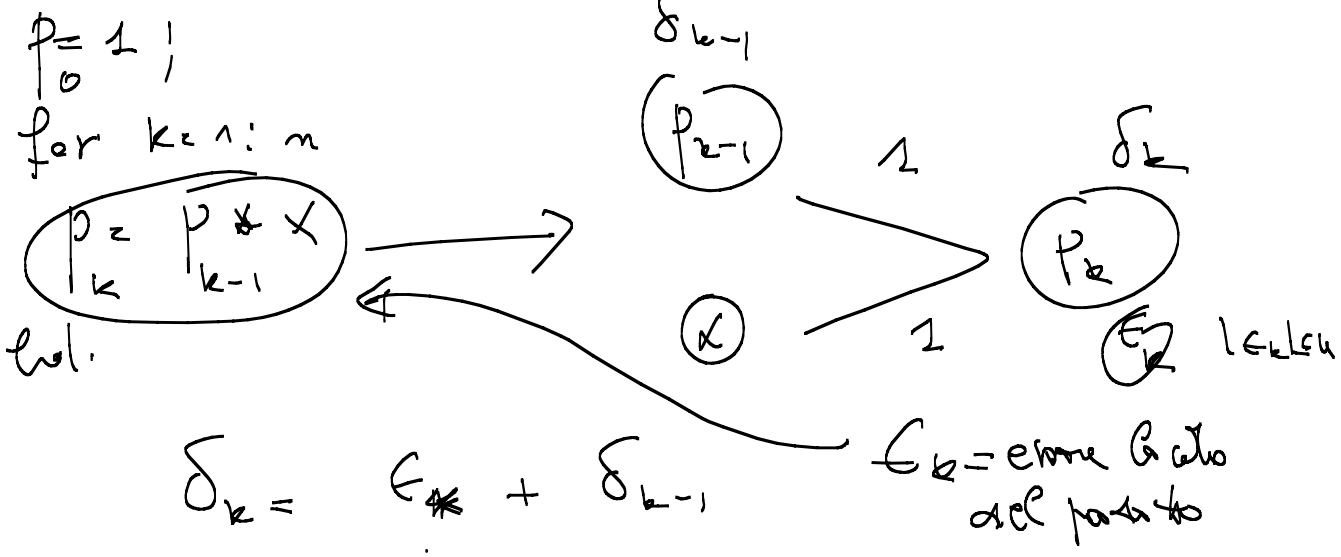
$$\doteq \frac{x^n\left(1+\sum_{i=1}^{n-1}\epsilon_i\right)-x^n}{x^n} = \frac{x^n\sum_{i=1}^{n-1}\epsilon_i}{x^n} = \sum_{i=1}^{n-1}\epsilon_i$$

$$|\epsilon_{alg}| \doteq \left|\sum_{i=1}^{n-1}\epsilon_i\right| \le \sum_{i=1}^{n-1}|\epsilon_i| \le (n-1)u$$

NUMERICAMENTE STABILE

$(C\epsilon m \simeq \epsilon_{alg})$

SCRIVERE ALGORITMICO $f(x) \in x^n$



$$\delta_k = \epsilon_k + \delta_{k-1}$$

$$|\delta_k| = |\epsilon_k + \delta_{k-1}| \le |\epsilon_k| + |\delta_{k-1}| \le$$

$$|\delta_{k-1}| + u \le |\delta_{k-2}| + 2u$$

$$\le |\delta_1| + (k-1)u$$

$$\le |\delta_0| + ku = ku$$

(se $|\epsilon_i| \ge 0 \rightarrow |\delta_k| \le (k-1)u$)

$|\delta_n| \leq (n+1)u \quad (|\delta_n| \leq nu)$

Algoritmo 2. $f(x) = e^{n \log(x)} = x^n$

Oss: funzione f è usata ma è calcolata mediante altre funzioni non usate.
$h(x) = e^x \quad g(x) = \log(x)$

Oss: Questa in generale non può essere ottenuta dall'analisi degli algoritmi con gli strumenti precedenti. Serve che qualcuno ci dica come è calcolata la funzione non usata

Oss: Aggiungiamo al modello ipotesi che

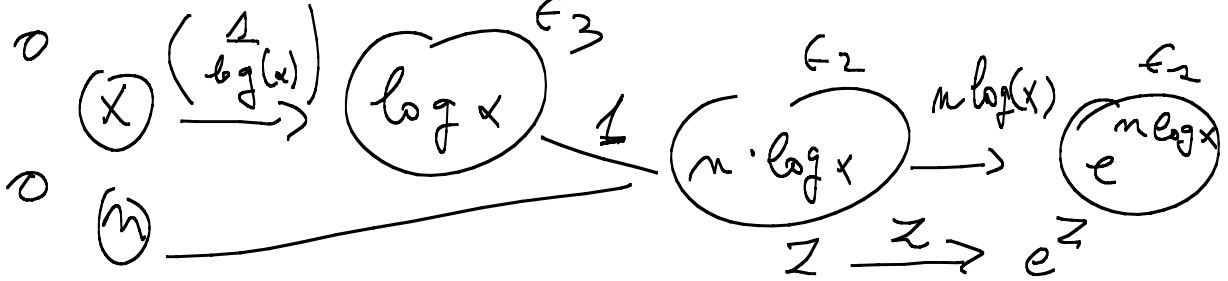
$$\widetilde{Exp}(x) = e^x \cdot (1+\epsilon_1) \quad |\epsilon_1| \leq u$$

$$\widetilde{Log}(x) = \log(x) \cdot (1+\epsilon_2) \quad |\epsilon_2| \leq u$$

$X \in F(B, t, m, M) \quad (|\epsilon_1|, |\epsilon_2| \leq \underline{\underline{k}} u)$

Oss: IN QUESTO CASO TRATTIAMO LE FUNZIONI LOGARITMICHE COME RAZIONALI

$\Rightarrow$ ANALISI DEL GRAFO



$|\epsilon_1| \leq u, \ |\epsilon_2| \leq u \ |\epsilon_3| \leq u$

$x \rightsquigarrow \log(x)$

$$C_x = \frac{\frac{1}{x}}{\log(x)} \, x = \frac{1}{\log(x)} \quad - \quad -$$

$x \rightsquigarrow e^x \qquad C_x = \frac{e^x}{e^x} \, x = x$

$$\sigma_{alg} = \epsilon_1 + n \log(x) (\epsilon_2 + \epsilon_3)$$

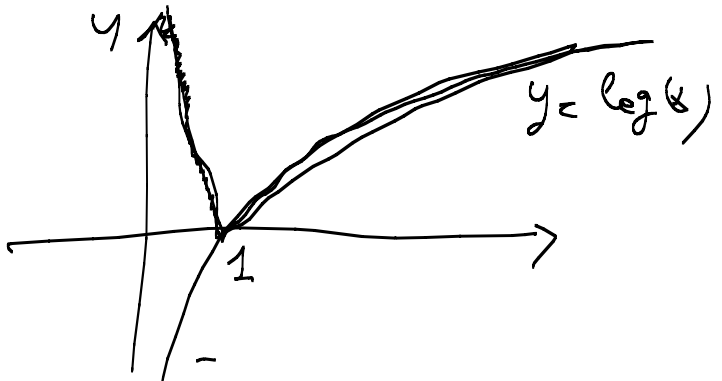
$$|\sigma_{alg}| = |\epsilon_1 + n \log(x) (\epsilon_2 + \epsilon_3)| \leq$$

$$|\epsilon_1| + |n \log(x) \cdot (\epsilon_2 + \epsilon_3)| \leq$$

$$u + |n \log(x)| \cdot |\epsilon_2 + \epsilon_3| \leq$$

$$u + n|\log(x)|\left(|\epsilon_2| + |\epsilon_3'|\right) \leq$$

$$u + 2u\, n|\log(x)| = u\cdot\left(1 + 2n|\log(x)|\right)$$



$$\log(x) = \log_e(x) = \ln(x)$$

ALGORITMO È NUMERICAMENTE INSTABILE QUANDO
X È PICCOLO O X È GRANDE

ANALISI DELLA STABILITÀ ⇒ ALG 1 È PREFERIBILE

COSTO CON PUNTATORE

ALG 1 $(n-1)$ operazioni moltiplicative
$O(n)$ operazioni moltiplicative
$\leq kn$

ALG 2 2 operazioni non moltiplicative!!
1 moltiplicazione

Esercizio: $n = 2^k$ $f(x) = x^n = x^{2^k}$

① determinare un algoritmo che valuta $f(x)$ con $k = \log_2(n)$ operazioni moltiplicative.

② Eseguo l'analisi dell'errore algoritmico per questo algoritmo

③ (Condizionamento ⟶ è indipendente dall'algoritmo di calcolo)

MATLAB

$f(x) = e^x$ ⟶ formula di Taylor

$$e^x \simeq 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

polinomio di Taylor

IMPLEMENTARE UN ALGORITMO CHE:

INPUT x, n

OUTPUT $y \leftarrow 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$

```
y = 1;
for k = 1:n
    y = y + x^(k)/factorial(k);
end
```

Costo computazionale al passo k $\simeq$ 2k prodotti.

$$C = 2 + 2\cdot 2 + 2\cdot 3 + \dots + 2n$$

$$= 2\cdot(1+2+3+\dots+n) = 2\cdot\frac{n\cdot(n+1)}{2} = O(n^2)$$

$$1+2+\dots+n = \frac{n\cdot(n+1)}{2}$$

Costo quadratico in n $\longrightarrow$ $T \simeq n^2 \cdot t_{op}$

```
y = 1; p = 1; m = 1
for k = 1: n
    p = p * x;        --> accumulo le potenze
    m = m * k;        --> accumulo il fatt.
  y = y + p/m;        --> accumulo la somma
end
```

Al passo k 4 operazioni aritmetiche / 3 operazioni rilevanti

(Alcune volte si afferma che $t_* \gg t_+$)

(Sì. costo booleano) $\longrightarrow$ Costo operazioni bit

Costo totale $C = 4n$ ops $(3n)$

$y = 1$; $z = 1$

for $k = 1 : n$

$z = z * x / k$; ← calcolo l'esplosione dei numeratori

$y = y + z$;

end

FINE

$k=1$ $z = \frac{x}{1}$; $k=2$ $z = \frac{x}{1} \cdot \frac{x}{2} = \frac{x^2}{2}$ $k=3$: $z = \frac{x^2}{2} \cdot \frac{x}{3} = \frac{x^3}{3!}$