Calcolo Numerico: Laboratorio
Lezione 2

Luca Gemignani <luca.gemignani@unipi.it>

05 Marzo 2020

1 Analisi dell'errore

Esercizio 1. Consideriamo il calcolo della funzione

$$f(x) = \frac{x-1}{x} = 1 - 1/x$$

In macchina calcoliamo

$$g_1(\tilde{x}) = (\tilde{x} \ominus 1) \oslash \tilde{x}; \quad g_2(\tilde{x}) = 1 \ominus (1/ \oslash \tilde{x}).$$

Valutiamo gli errori

$$\epsilon_{TOT_1} = \frac{g_1(\tilde{x}) - f(x)}{f(x)}; \quad \epsilon_{TOT_2} = \frac{g_2(\tilde{x}) - f(x)}{f(x)}.$$

Ricordiamo che

$$\epsilon_{TOT} \doteq \epsilon_{IN} + \epsilon_{ALG}.$$

Studiamo i due contributi separatamente.

$$\epsilon_{IN} \doteq \frac{f'(x)}{f(x)} x\epsilon_x = \frac{1/x^2}{(x-1)/x} x\epsilon_x = \frac{1}{x-1}\epsilon_x$$

da cui il problema è mal condizionato intorno a 1. Per l'errore algoritmico nel calcolo di $g_1(\tilde{x})$ si ricava dal grafo associato

$$\epsilon_{ALG_1} = \epsilon_1 + \epsilon_2$$

da cui $|\epsilon_{ALG_1}| \leq 2u$ e quindi il primo algoritmo è numericamente stabile. Per l'errore algoritmico nel calcolo di $g_2(\tilde{x})$ si ricava dal grafo associato

$$\epsilon_{ALG_2} = \epsilon_2 - \frac{1}{x-1}\epsilon_1$$

da cui

$$|\epsilon_{ALG_2}| \leq (1 + 1/|x-1|)u$$

e quindi il secondo algoritmo è numericamente instabile in un intorno di 1.

Esercizio 2. Si consideri il calcolo della funzione $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$ con $x_i > 0$ mediante il seguente algoritmo:

```
s=x(1);
for k=2:n
s=s+x(k);
end
```

Valutare il condizionamento del problema e la stabilità dell'algoritmo di calcolo.

Dalla generalizzazione della formula per l'errore inerente si ottiene

$$\epsilon_{IN} \doteq (1/f(\boldsymbol{x})) \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\boldsymbol{x}) x_i \epsilon_i, \quad \boldsymbol{x} = [x_1, \ldots, x_n]$$

da cui

$$\epsilon_{IN} \doteq \sum_{i=1}^{n} \frac{x_i}{x_1 + \ldots + x_n} \epsilon_i$$

e quindi

$$|\epsilon_{IN}| \leq u.$$

IL problema è pertanto ben condizionato. Per l'errore algoritmico posto x_i numeri di macchina e $s = s_k = x_1 + \ldots + x_k$ il valore della variabile s al tempo $k-1$ si avrà in macchina

$$\tilde{s}_k = s_k(1 + \delta_k), \quad 1 \leq\leq n, \quad \delta_1 = 0.$$

Inoltre dall'analisi dell'errore algorimico mediante il grafo associato si ricava

$$\delta_{k+1} = \epsilon_{k+1} + (s_k/s_{k+1})\delta_k, \quad |\epsilon_{k+1}| \leq u.$$

Segue che

$$|\delta_{k+1}| \leq u + \delta_k \leq \ldots \leq ku.$$

In particolare

$$|\epsilon_{ALG}| = |\delta_n| \leq (n-1)u$$

e l'algoritmo risulta numericamente stabile.

Esercizio 3. Si osservi che il valore dei coefficienti di amplificazione s_k/s_{k+1} dipende dall'ordinamento degli addendi. Intuitivamente quale ordinamento risulta preferibile?

2 Iniziare a Programmare in MatLab

Iniziamo con qualche semplice esempio.

```
function f=fact(n);
% calcola il fattoriale di n
% n dev'essere un intero
f=1;
```

```
% f fa da accumulatore: parte da 1,
% a ogni passo, lo moltiplico per k
for k=1:n
  f=f*k;
end
% ora f vale n!
end
```

Va scritto in un file chiamato `fact.m` e messo *nella cartella da cui abbiamo lanciato Matlab*; poi possiamo lanciarlo:

```
>> fact(10)
ans = 3628800
```

Esercizio 4. Scrivete una funzione `pow(x,n)` che calcoli x^n, per $n \geq 1$.

3 Calcolo dell'esponenziale

La formula di Taylor permette di esprimere una funzione come un polinomio con infiniti termini, e troncando in modo opportuno questa serie è possibile approssimare una funzione con un polinomio.

In particolare, data una funzione $f \in C^n$, cioè derivabile n-volte con derivata n-esima continua, tale che $f : (a, b) \to \mathbf{R}$. Preso un punto $x_0 \in (a, b)$, abbiamo che

$$f(x) = f(x_0) + f'(x_0)(x - x_0)| + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x)$$

dove $R_n(x)$ è detto resto di Taylor e nella sua forma di Lagrange è dato da

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1},$$

sotto l'ipotesi che $f^{(n+1)}(x)$ esista per ogni $x \in (a, b), x \neq x_0$, e con $|\xi - x_0| < |x - x_0|$.

Si può semplicemente vedere che il polinomio di Taylor per $x_0 = 0$, arrestato al termine n-esimo della funzione esponenziale è

$$1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

da cui possiamo scrivere la seguente funzione per approssimare il valore dell'esponenziale in un punto x.

```
function a=myexp(x,n)
% calcola exp(x) con la serie di Taylor troncata all'n-esimo termine
  a=1; %accumulatore
  for k=1:n
    a=a+pow(x,k)/fact(k);
  end
end
```

Qualche esperimento su quanti termini servono per approssimare bene — confrontiamo con la funzione `exp(x)` di Matlab, che calcola l'esponenziale meglio di noi.

```
>> myexp(1,5)
ans = 2.7167
>> myexp(10,50)
ans = 2.2026e+04
>> exp(10)
ans = 2.2026e+04
>> format long
>> myexp(10,50)
ans = 22026.4657948067
>> exp(10)
ans = 22026.4657948067
```

Due problemi:

1. Inaccurato: prova `myexp(-20,500)`
2. Lento: con n termini, il numero di operazioni da eseguire cresce come n^2 (perché?)

Risolviamo il problema 2. introducendo un altro accumulatore:

```
function a=myexp2(x,n)
  %calcola e^x con Taylor troncato
  %ma usa solo O(n) operazioni
  t=1; %accumulatore che contiene il termine generico della sommatoria
  a=1; %accumulatore che contiene le somme parziali
  for k=1:n
    t=t*x/k;
    a=a+t;
  end
end
```

Ora va meglio:

```
>> myexp(-20,500)
ans = NaN
>> myexp2(-20,500)
ans = 5.62188447213042e-09
```

Cosa succedeva?

```
>> fact(500)
ans = Inf
>> pow(-20,500)
ans = Inf
>> Inf/Inf
ans = NaN
```

Ci sono ancora pesanti perdite di accuratezza sui numeri negativi:

```
>> myexp2(-30,500)
ans = -3.06681235635622e-05
```

Un esponenziale dovrebbe sempre essere positivo... Ci sono pesanti errori di cancellazione nella formula che abbiamo usato (perché?) La soluzione: cambiare algoritmo e sceglierne uno che non porti a errori di cancellazione che sono dovuti al fatto che per $x < 0$ la serie esponenziale è a segni alterni.

```
>> exp(-30)
ans = 9.3576e-14
>> myexp2(-30,500)
ans = -3.0668e-05
>> 1/myexp2(30,500)
ans = 9.3576e-14
>> format long
>> exp(-30)
ans = 9.35762296884017e-14
>> 1/myexp2(30,500)
ans = 9.35762296884017e-14
```