CALCOLO NUMERICO
(A.A. 2012-2013)

Prof. F. Pitolli

Appunti della prima lezione

Obiettivi Orizzonte 2020 (Horizon 2020)

La Commissione Europea ha presentato la proposta per il *programma di ricerca e innovazione 2014-20* che prenderà il nome di **Orizzonte 2020**.

I finanziamenti sono stati organizzati su **3 obiettivi strategici**:

1) 24,6 miliardi di euro per **Excellent Science**, destinati a garantire il primato dell'Europa nel settore scientifico a livello mondiale

2) 17,9 miliardi di euro per **Industrial Leadership** rivolti a sostenere la ricerca e l'innovazione dell'industria europea

3) 31,7 miliardi di euro per **Societal Challenges**. Risorse destinate ad affrontare le grandi sfide globali nei settori:

- sanità , cambiamenti demografici e benessere;
- sicurezza alimentare, agricoltura sostenibile, ricerca marina e marittima e bioeconomia;
- energia da fonti sicure, pulita ed efficiente;
- trasporti intelligenti, ecologici e integrati;
- azione per il clima, efficienza sotto il profilo delle risorse e materie prime;
- società inclusive, innovative e sicure.

Lo **sviluppo sostenibile** sarà un obiettivo generale di Orizzonte 2020: almeno il 60% della dotazione complessiva di Orizzonte 2020 sarà collegata allo sviluppo sostenibile e, in particolare, si prevede che il 35% circa del bilancio di Orizzonte 2020 sarà costituito da spese connesse con il **clima**.

Cercare sul web *smart city, nanotechnology, neurosciences, ...*

Cosa è il CALCOLO NUMERICO ?

È quella branca della **matematica** che

costruisce e **analizza**
i **metodi numerici**
adatti a risolvere, con l'aiuto del **calcolatore**,
differenti **problemi matematici**
che nascono in varie discipline:
ingegneria, **economia**, **biologia**, **medicina** ...

Cercare sul web immagini relative a **'calcolo numerico'**

Problema da risolvere

Esempio: Calcolare la temperatura di un gas noti la **pressione**, il **volume occupato** e il **numero di moli**

↓

Schematizzazione sulla base di ipotesi esemplificative → **errori inerenti**

↓

Modello matematico

Esempio: **Legge dei gas ideali**: $PV = NRT$

P: pressione, V: volume, T: temperatura

R: costante

Esempio

Supponiamo di effettuare un esperimento in cui vengono **misurati** la **pressione**, il **volume** e il **numero di moli** e si vuole **predire la temperatura**.

$P = 1.00\,\text{atm}$ $\quad V = 0.100\,\text{m}^3$
$N = 0.00420\,\text{mol}$ $\quad R = 0.08206$

$$T = \frac{PV}{NR} = \frac{(1.00)(0.100)}{(0.00420)(0.08206)} = 290.15\,\text{K} = 17^o\,\text{C}$$

La temperatura **misurata** è 15^o C. Se si ripete l'esperimento con gli stessi valori di N e R, ma raddoppiando la pressione e dimezzando il volume, la temperatura predetta è ancora 17^o C mentre quella misurata è 19^o C.

Quale dei due risultati è corretto?

Esempio

Assumiamo che gli **errori sui dati** siano

$\Delta P = 0.5 \cdot 10^{-3}$ $\quad \Delta V = 0.5 \cdot 10^{-4}$

$\Delta N = 0.5 \cdot 10^{-6}$ $\quad \Delta R = 0.5 \cdot 10^{-6}$

Allora per la temperatura **predetta** dalla legge dei gas perfetti si ha:

$$\frac{PV-(V\Delta P+P\Delta V)}{NR+(R\Delta N+N\Delta R)} \le T \le \frac{PV+(V\Delta P+P\Delta V)}{NR-(R\Delta N+N\Delta R)}$$

Quindi l'**errore propagato** sulla temperatura predetta è

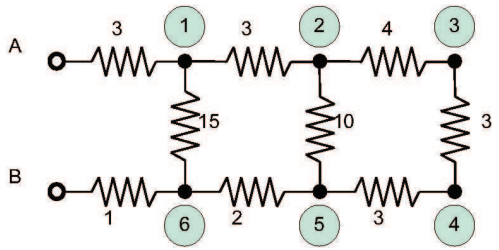
$286.53\,\text{K} \le T \le 293.78\,\text{K}$ (primo esperimento)

$285.81\,\text{K} \le T \le 294.51\,\text{K}$ (secondo esperimento)

Poiché $15^o\,\text{C} = 288.16\,\text{K}$ e $19^o\,\text{C} = 292.16\,\text{K}$, entrambi i valori misurati sono compresi nell'intervallo dei valori predetti.

Esempio: circuito elettrico

Calcolare dei potenziali v_1, v_2, ..., v_6 nei nodi del circuito



(i valori delle resistenze sono date in *Ohm*) quando tra A e B è applicata una differenza di potenziale di 100 *Volt*.

Esempio: circuito elettrico

Nodo 1: $I_{A1} + I_{21} + I_{61} = \frac{100-v_1}{3} + \frac{v_2-v_1}{3} + \frac{v_6-v_1}{15} = 0$

Applicando la Legge di Kirchoff a ciascun nodo si ottiene il **sistema lineare**:

$$\begin{cases} 11v_1 - 5v_2 - v_6 = 500 \\ -20v_1 + 41v_2 - 15v_3 - 6v_5 = 0 \\ -3v_2 + 7v_3 - 4v_4 = 0 \\ -v_3 + 2v_4 - v_5 = 0 \\ -10v_4 + 28v_5 - 15v_6 = 0 \\ -2v_1 - 15v_5 + 47v_6 = 0 \end{cases}$$

Altri esempi di modelli matematici possono essere i **sistemi di equazioni non lineari** (es., equilibri chimici, ottimizzazione), gli **integrali** (es., aree, volumi, energia), i **sistemi di equazioni differenziali** (es., sistemi dinamici).

↓

Metodo numerico → **errori di troncamento**

(la scelta è un *arte*) **Esempio**: per i sistemi lineari
metodo di Cramer,
metodo di eliminazione di Gauss
metodo di Jacobi

↓

Algoritmo → **stabilità**

↓

Soluzione numerica → **errori di arrotondamento**

La soluzione numerica è **accettabile** solo se si sanno **stimare** gli **errori** da cui è affetta.

Errori di arrotondamento - 1

Il **sistema di numeri** disponibile su un **calcolatore** è piuttosto primitivo: è un **sistema finito** di **numeri di lunghezza finita**, mentre l'**analisi matematica** o la **geometria** hanno a che fare con **numeri infiniti** di **lunghezza infinita**.

Analisi Matematica — Analisi Numerica

Geometria, Algebra

↓ ↓

$\mathbb{R}$: *Numeri reali* → $\mathbb{F}$: *Numeri macchina*

Errori di arrotondamento

Errori di arrotondamento - 2

Analisi Matematica: $\pi = 3.14159265358979323846264 33...$

$\sqrt{2} = 1.41421356237309504880168 87...$

Analisi Numerica:

```
atan(1)*4  →  3.14159265358979
sqrt(2)    →  1.41421356237310
```

L' **arrotondamento** è la prima fonte di **errore**: i dati di **input**, che hanno in generale un **numero infinito** di cifre, vengono trasformati dal calcolatore, tramite **arrotondamento**, in **numeri macchina**, cioè numeri con un **numero finito** di cifre.

Errori di arrotondamento: esempi

Numeri reali
3.751294196 5...
...
3.751294196 6...
...
3.751294196 7...
...
3.751294196 8...
...
3.751294196 9...
...
3.751294197 0...
...
3.751294197 1...
...
3.751294197 2...
...
3.751294197 3...
...
3.751294197 4...

⇒ Arrotondamento

Numeri macchina
3.751294197

Errori di arrotondamento - 3

Errore di arrotondamento = **Numero reale** - **Numero macchina**

Numeri reali	*Errori di arrotondamento*
3.7512941965...	$+0.5... \cdot 10^{-9}$
3.7512941966...	$+0.4... \cdot 10^{-9}$
3.7512941967...	$+0.3... \cdot 10^{-9}$
3.7512941968...	$+0.2... \cdot 10^{-9}$
3.7512941969...	$+0.1... \cdot 10^{-9}$
3.7512941970...	$+0.0... \cdot 10^{-9}$
3.7512941971...	$-0.1... \cdot 10^{-9}$
3.7512941972...	$-0.2... \cdot 10^{-9}$
3.7512941973...	$-0.3... \cdot 10^{-9}$
3.7512941974...	$-0.4... \cdot 10^{-9}$

$\Rightarrow \quad |\textit{Errore di arrotondamento}| \leq 0.5 \cdot 10^{-9}$

Se i numeri macchina sono **arrotondati** alla $\mathrm{D} - esima$ cifra **decimale** $\Rightarrow$ l'**errore di arrotondamento** è compreso nell'intervallo

$$\left[-0.5 \cdot 10^{-\mathrm{D}}, +0.5 \cdot 10^{-\mathrm{D}}\right]$$

Errori di arrotondamento: esempi

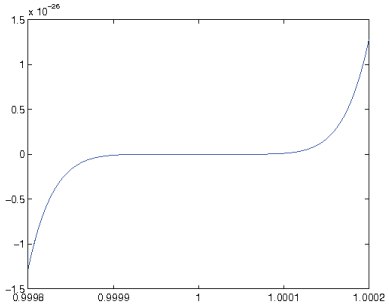
$$q_1(x) = (x-1)^7 \quad \longleftrightarrow \quad q_2(x) = x^7 - 7x^6 + 21x^5 - \\ -35x^4 + 35x^3 - 21x^2 + 7x - 1$$

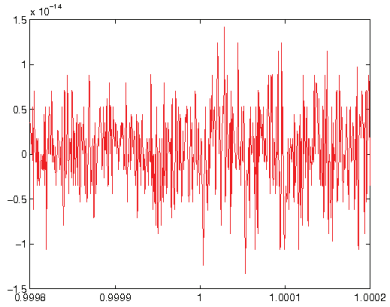
Dal punto di vista dell'**algebra** le quantità $q_1(x)$ e $q_2(x)$ sono **identiche**. Calcoliamo $q_1(x)$ e $q_2(x)$ **numericamente** nell'intervallo $[0.9998, 1.0002]$ utilizzando una **calcolatrice** che lavora con **10 cifre significative**.

x	$q_1(x)$	$q_2(x)$	**Valore esatto**	**Errore di arrotondamento**
1	0	0	0	0
1.0001	10^{-28}	-10^{-10}	10^{-28}	$\simeq -10^{-10}$
...	...	...	...	...

Esercizio (gnuplot)

Calcolare $q_1(x)$ e $q_2(x)$ **numericamente** nell'intervallo $[0.9998, 1.0002]$ utilizzando il calcolatore.



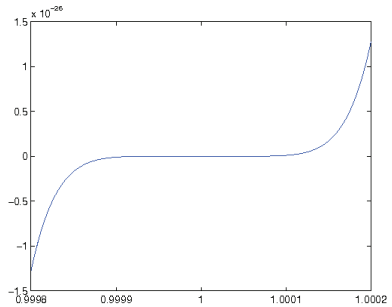


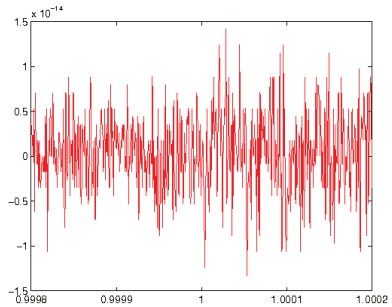

```
gnuploplot> set xrange [0.9998:1.0002]
gnuploplot> plot (x-1)**7
gnuploplot> plot x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x-1
```

Nota: Il **C** e il **Fortran** lavorano con **7 cifre significative** in **singola precisione** e con **15 cifre significative** in **doppia precisione**.

Esercizio (Matlab)

Calcolare $q_1(x)$ e $q_2(x)$ **numericamente** nell'intervallo $[0.9998, 1.0002]$ utilizzando il calcolatore.






```
figure(1); fplot('(x-1)^7',[0.9998,1.0002],'b')
figure(2); fplot('x^7-7*x^6+21*x^5-35*x^4+35*x^3-21*x^2+7*x-1',[0.9998,1.0002],'r')
```

Nota: **MATLAB** lavora sempre con **15 cifre significative**.

Rappresentazione dei numeri

Un numero reale x è rappresentato nel calcolatore come un numero macchina (*numero floating-point*)

$fl(x) = (-1)^s \cdot (0.a_1 a_2 \ldots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}$ $\quad a_1 \neq 0$

s	$= 0, 1$:	**segno**
β	(intero ≥ 2):	**base**
m	(intero di lunghezza t):	**mantissa**
e	(intero):	**esponente**

In **MATLAB**: $\beta = 2$, $t = 53$, $-1021 \leq e \leq 1024$.

Errore di arrotondamento: $\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\varepsilon$

$\varepsilon = \beta^{1-t} = 2^{-52}$ `>> eps` `2.220446049250313e-016`

Nota. 53 cifre significative in base 2 corrispondono a **15 cifre significative** in base 10.

Esempio (Matlab)

153/7 = 21.857142857142857142857142857...

```
>> format short      21.8571
>> format short e    2.1857e+001
>> format short g    21.857
>> format long       21.85714285714286
>> format long e     2.185714285714286e+001
>> format long g     21.8571428571429
```

Nota. Nel formato `short` vengono mostrate solo 6 cifre significative mentre nel formato `long` vengono mostrate tutte le cifre significative. I calcoli vengono comunque fatti utilizzando **tutte** le cifre significative.

Underflow e overflow (Matlab)

Poiché $-1021 \leq e \leq 1024$, non si possono rappresentare numeri con valore assoluto **inferiore** a $x_{min} = \beta^{-1022}$

```
>> realmin      2.225073858507201e-308
```

e **superiore** a $x_{max} = \beta^{1024}(1 - \beta^{-t})$

```
>> realmax      1.797693134862316e+308
```

Nota. Un numero **più piccolo** di x_{min} viene trattato come 0 (**underflow**). Un numero **più grande** di x_{max} produce un messaggio di **overflow** e viene memorizzato in una variabile `Inf`.

Esercizi Matlab

Che output producono le instruzioni seguenti?

```
>> a=1;b=1;while a+b~=a;b=b/2;[a b],end
```

```
>> a=1;b=1;while a+b~=a;b=b*2;[a b],end
```

```
>> x=1.e-15;((1+x)-1)/x
```

```
>> x=0;sin(x)/x
```

Tipi di dati elementari (Programma C)

```
#include <stdio.h>
#include <limits.h>
#include <float.h>

main(){

  printf("INTERI dim. %d\n",sizeof(int));
  printf("minimo  %d  ",INT_MIN);
  printf("massimo %d\n",INT_MAX);

  printf("\nFLOAT dim. %d\n",sizeof(float));
  printf("Cifre di precisione %d\n",FLT_DIG);
  printf("minimo  esp. %d  ",FLT_MIN_10_EXP);
  printf("massimo esp. %d\n",FLT_MAX_10_EXP);

  printf("\nDOUBLE dim. %d\n",sizeof(double));
  printf("Cifre di precisione %d\n",DBL_DIG);
  printf("minimo  esp. %d  ",DBL_MIN_10_EXP);
  printf("massimo esp. %d\n",DBL_MAX_10_EXP);
```



```
  printf("\nLONG DOUBLE dim. %d\n",sizeof(long double));
  printf("Cifre di precisione %d\n",LDBL_DIG);
  printf("minimo  esp. %d  ",LDBL_MIN_10_EXP);
  printf("massimo esp. %d\n",LDBL_MAX_10_EXP);

}
```

Output (dipende dal computer)

```
INTERI dim. 4
minimo  -2147483648  massimo 2147483647

FLOAT dim. 4
Cifre di precisione 6
minimo  esp. -37  massimo esp. 38

DOUBLE dim. 8
Cifre di precisione 15
minimo  esp. -307  massimo esp. 308

LONG DOUBLE dim. 12
Cifre di precisione 18
minimo  esp. -4931  massimo esp. 4932
```


Cancellazione numerica

Consideriamo l'**equazione di secondo grado**

$$ax^2 + bx + c = 0$$

Dall'**algebra** sappiamo che se $\Delta = b^2 - 4ac > 0$, l'equazione ha 2 **soluzioni reali** distinte:

$$x_1 = \frac{-b-\sqrt{\Delta}}{2a} \qquad x_2 = \frac{-b+\sqrt{\Delta}}{2a}$$

Calcoliamo x_1 e x_2 **numericamente** con la **calcolatrice**.

a, b, c	x_1	x_2	$ax_1^2 + bx_1 + c$	$ax_2^2 + bx_2 + c$
1 4 3	-3	-1	0	0
1 −206.5 0.01021	$4.945 \cdot 10^{-5}$	206.4999506	$-1.42 \cdot 10^{-6}$	$8.9 \cdot 10^{-6}$

Calcoliamo ora le soluzioni con le formule

$$x_1 = \frac{2c}{-b+\sqrt{\Delta}}, \quad x_2 = \frac{c}{ax_1}.$$

a, b, c	x_1	x_2	$ax_1^2 + bx_1 + c$	$ax_2^2 + bx_2 + c$
1 4 3	-3	-1	0	0
1 −206.5 0.01021	$4.9443111 \cdot 10^{-5}$	206.499951	0 (*macchina*)	$9.66 \cdot 10^{-5}$

Esercizio. Ripetere il calcolo delle radici con la propria calcolatrice e con il calcolatore. Confrontare i risultati ottenuti con quelli dati nelle tabelle.

Cosa è successo? Per calcolare x_2 bisogna calcolare la quantità $-b - \sqrt{\Delta}$.

Primo caso: $a = 1, b = 4, c = 3$

$\rightarrow \sqrt{\Delta} = 2$

Secondo caso: $a = 1, b = -206.5, c = 0.01021$

$\rightarrow \sqrt{\Delta} = 206.4999011...$

In questa caso b è **negativo**, quindi bisogna calcolare la **differenza** tra due numeri molto vicini $\rightarrow$ **cancellazione numerica**.

Script MATLAB

```
% Calcolo delle radici di un'equazione di II grado
%
% Dati di input
%
a = input('a: ');
b = input('b: ');
c = input('c: ');
%
% Calcolo del discriminante
%
Delta = b^2-4*a*c;
fprintf('Delta = %-10.6g\n',Delta)
%
% Calcolo e stampa delle radici (formule standard)
%
x1 = -(b+sqrt(Delta))/(2*a);
x2 = -(b-sqrt(Delta))/(2*a);
fprintf('Radici (formule standard)\n')
fprintf('x1 = %-19.17g\t\t\t x2 = %-19.17g\n',x1,x2)
%
% Calcolo e stampa di a*x^2+b*x+c (formule standard)
%
p1 = a*x1^2+b*x1+c;
p2 = a*x2^2+b*x2+c;
fprintf('a*x1^2+b*x1+c = %-15.10g\t a*x2^2+b*x2+c = %-15.10g\n',p1,p2)
```



```
%
% Calcolo e stampa delle radici (formule modificate)
%
x1 = 2*c/(-b+sqrt(Delta));
x2 = c/(a*x1);
fprintf('Radici (formule modificate)\n')
fprintf('x1 = %-19.17g\t\t\t x2 = %-19.17g\n',x1,x2)
%
% Calcolo e stampa di a*x^2+b*x+c (formule standard)
%
p1 = a*x1^2+b*x1+c;
p2 = a*x2^2+b*x2+c;
fprintf('a*x1^2+b*x1+c = %-15.10g\t a*x2^2+b*x2+c = %-15.10g\n',p1,p2)
```


Programma C

```
/*
 * programma: soleq2.c
 * Soluzione di un'equazione di secondo grado ax^2 + bx + c = 0
 *
 * Compilazione:
 * gcc -g -Wall -o soleq2 soleq2.c -lm
 *
 * Input:
 *  double a,b,c: coefficienti dell'equazione
 * Variabili:
 *  double disc: discriminante b^2-4ac
 * Output:
 *  double x1, x2: soluzioni dell'equazione (La forma A  quella
 *  "classica"; la forma B  quella "modificata").
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
double a, b, c;
double disc, x1, x2;
double p1, p2;
```

```
printf("Size of double = %ld bytes\n", sizeof(double));

/* Inizializzazione dati */
a = b = c = 0.0;
disc = x1 = x2 = 0.0;

/* Lettura dati input */
printf("Inserisci a, b, c: ");
scanf("%lf%lf%lf", &a, &b, &c);

/* Calcolo del discriminante */
disc = b * b - 4.0 * a * c;

/* Calcolo della soluzione a seconda del
 * valore del discriminante */
if(disc < 0)
{
printf("La soluzione non esiste\n");
} /* fine costrutto if */
else if(disc == 0.0)
{
x1 = -b/(2.0 * a);
printf("Le soluzioni sono coincidenti\n");
printf("x1 = x2 = %lf\n", x1);
} /* fine costrutto  else if */
else
```



```
{
/* forma A */
x1 = (-b - sqrt(disc))/(2.0 * a);
x2 = (-b + sqrt(disc))/(2.0 * a);
printf("Le soluzioni (forma A) sono: ");
printf("x1 = % 19.17lg;  x2 = % 19.17lg\n", x1, x2);
/* Calcolo e stampa di ax^2+bx+c (forma A) */
p1 = a*pow(x1, 2.0) + b*x1 + c;
p2 = a*pow(x2, 2.0) + b*x2 + c;
printf("ax1^2+bx1+c = % 15.10lg\t ax2^2+bx2+c = % 15.10lg\n", p1, p2);

/* forma B */
x1 = (2.0 * c)/(-b + sqrt(disc));
x2 = c/(a*x1);
printf("Le soluzioni (forma B) sono: ");
printf("x1 = % 19.17lg;  x2 = % 19.17lg\n", x1, x2);
/* Calcolo e stampa di ax^2+bx+c (forma A) */
p1 = a*pow(x1, 2.0) + b*x1 + c;
p2 = a*pow(x2, 2.0) + b*x2 + c;
printf("ax1^2+bx1+c = % 15.10lg\t ax2^2+bx2+c = % 15.10lg\n", p1, p2);
} /* fine costrutto  else */

return 0;
} /* fine funzione main */
```


Programma FORTRAN

```
      program solequ2
*
* Soluzione di un'equazione di secondo grado: ax^2+bx+c=0.
*
* Input:
* - real a,b,c: coefficienti dell'equazione
* Variabili:
* - real disc: discriminante b^2-4ac
* Output:
* - real x1,x2: soluzioni dell'equazione (la forma A e' quella
*               'classica'; la forma B e' quella 'modificata').
*
      implicit none
*
      real a, b, c
      real disc, x1, x2
*
* Lettura dei dati di input
*
      write (*,*) 'Inserisci a, b, c:'
      read (*,*) a, b, c
*
* Calcolo del discriminante
*
      disc = b*b - 4.*a*c
```



```
*
* Calcolo della soluzione a seconda del valore del discriminante
*
      if ( disc .lt. 0. ) then
        write (*,*) 'La soluzione non esiste'
      elseif (disc .eq. 0. ) then
        x1 = - b/(2.*a)
        write (*,*) 'Le soluzioni sono coincidenti: '
        write (*,*) 'x1=', x1
      else
        x1 =  (-b+sqrt(disc))/(2.*a)
        x2 =  (-b-sqrt(disc))/(2.*a)
        write (*,*) 'Le soluzioni (forma A) sono: x1=',
&                   x1, ' x2=', x2
        x1 = -2.*c/(b+sqrt(disc))
        x2 = c/(a*x1)
        write (*,*) 'Le soluzioni (forma B) sono: x1=',
&                   x1, ' x2=', x2
      endif
*
* Fine del programma
*
      stop
      end
```

Algoritmo

L'**algoritmo** è una successione di **istruzioni**, **finita** e **non ambigua**, che consente di ottenere risultati numerici a partire dai dati di input.

L'algoritmo viene implementato su calcolatore tramite un **linguaggio di programmazione**.

Le **istruzioni** sono **operazioni logiche** o **operazioni aritmetiche** date seguendo la **sintassi** del linguaggio di programmazione scelto.

Stabilità di un algoritmo

Anche se l'**errore di arrotondamento** è **"piccolo"**, la sua **propagazione** attraverso i calcoli può avere effetti **disastrosi**. Gli errori di arrotondamento possono venire **amplificati** durante i calcoli così da rendere la soluzione numerica del tutto **inaffidabile**. In questo caso si dice che l'**algoritmo** è **instabile**.

Se gli errori di arrotondamento **non** vengono **amplificati** durante i calcoli si dice che l' **algoritmo** è **stabile**.

Stabilità di un algoritmo: esempi

Modello matematico: $I_n = \frac{1}{e}\int_0^1 x^n e^x \, dx$

Tramite integrazione per parti si ottiene

$$I_n = \frac{1}{e}\left(e - \int_0^1 n x^{n-1} e^x \, dx\right) = 1 - nI_{n-1}$$

e continuando ...

$$\begin{aligned} I_n &= 1 - nI_{n-1} = 1 - n(1-(n-1)I_{n-2}) = \\ &= 1 - n + n(n-1)(1-(n-2)I_{n-3}) = \cdots = \\ &= 1 + \sum_{k=1}^{n-1} (-1)^k n(n-1)\cdots(n-k+1) + (-1)^n n! I_0 \end{aligned}$$

↑ Algoritmo

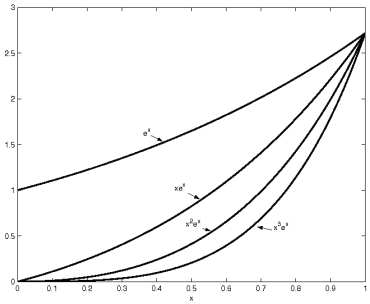
dove $I_0 = \frac{1}{e}\int_0^1 e^x \, dx = 1 - \frac{1}{e}$

$I_0 = 0.63212055882856 \rightarrow$ Numero macchina (14 cifre significative)

$I_1 = 1 - I_0 = 0.36787944117144$

$I_2 = 1-2+2!I_0 = -2+2I_0 = 0.26424111765712$

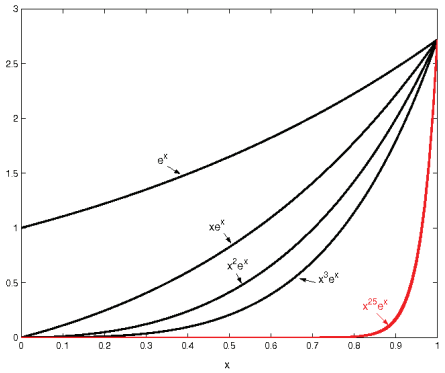
$I_3 = 1-3+3\cdot 2-3!I_0 = 4-6I_0 = 0.20727664702865$



$I_{25} = 0$

$I_{26} = -3.435973836800000e + 010$

Non è possibile!!



Script MATLAB

```
% Calcolo dell'integrale In
%
% Input
%
n = input('n: ');
%
% Dati
%
I0 = 1-1/exp(1)
%
% Calcolo e stampa dell'integrale
%
somma = 0;
for k=1:n-1
    somma = somma + (-1)^k*factorial(n)/factorial(n-k);
end
In = 1 + somma + (-1)^n*factorial(n)*I0;
fprintf('In = %-19.17g\n',In)
```

Nota. In alternativa, la sommatoria può essere calcolata utilizzando le **istruzioni vettoriali** di Matlab che sono più **efficienti** dell'istruzione `for`

```
k = 1:n-1;
somma=(-1).^k.*factorial(n)./factorial(n-k);
In = 1 + sum(somma) + (-1)^n*factorial(n)*I0;
```


Programma C

```
/*
 * programma: calcIn.c
 * Calcolo dell'integrale In con formula ricorsiva
 *
 * Compilazione:
 * gcc -g -Wall -o calcIn calcIn.c -lm
 *
 * Input:
 *  int n: potenza di x
 *  double I0: valore iniziale
 * Variabili:
 *  double somma: variabile di accumulazione
 * Output:
 *  double In: valore dell'integrale
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
int i, k, n;
double I0, In;
double somma, fatn, prodnk;
```



```
/* Inizializzazione dati */
i = k = n = 0;
I0 = In = 0.0;
somma = fatn = prodnk = 0.0;

/* Lettura dati input */
printf("Inserisci n: ");
scanf("%d", &n);

/* Dati */
I0 = 1.0 - 1.0/exp(1);

/* Calcolo del fattoriale di n */
fatn = 1.0;
for(i=2; i <= n; i++)
fatn = fatn * (double)i;
```

```
/* Inizio ciclo iterativo per il calcolo della sommatoria */
somma = 0.0;

for(k=1; k <= n - 1; k++)
{
/* Calcolo del prodotto n(n-1)...(n-k+1) */
prodnk = (double)n;
for(i = n - 1; i >= n - k + 1; i--)
{
prodnk = prodnk * (double)i;
} /* fine costrutto for i */

/* Sommatoria */
somma = somma + pow(-1.0, k) * prodnk;
} /* fine costrutto for k */

/* Calcolo dell'integrale */
In = 1.0 + somma + pow(-1.0, n) * fatn * I0;

/* Stampa dell'integrale */
printf("Valore dell'integrale: %lf\n", In);

return 0;
} /* fine funzione main */
```


Programma FORTRAN

```
      program CalcIn
*
* Calcolo dell'integrale In con formula ricorsiva
*
* Input:
*-integer n: potenza di x
*-I0: valore iniziale
* Variabili:
* - real somma: variabile di accumulazione
* Output:
* - real In: valore dell'integrale
*
      implicit none
*
      integer n, i, k
      double precision I0, In
      double precision somma, fatn, prodnk
*
* Lettura dei dati di input
*
      write (*,*) 'Inserisci n:'
      read (*,*) n
```



```
* Dati
*
      I0 = 1.-1./exp(1.)
*
* Calcolo del fattoriale n!
*
         fatn = 1
         do i = 2, n
           fatn = fatn *float(i)
         enddo
*
* Inizio ciclo iterativo per il calcolo della sommatoria
*
      somma = 0.;
      do k=1,n-1
*
* Calcolo del prodotto n(n-1)...(n-k+1)
*
         prodnk = float(n)
         do i = n-1, n-k+1, -1
           prodnk = prodnk * float(i)
         enddo
*
* Sommatoria
*
         somma = somma + (-1)**k * prodnk;
*
      enddo
```



```
*
* Calcolo dell'integrale
*
      In = 1. + somma + (-1)**n * fatn * I0;
*
* Stampa dell'integrale
*
      write (*,*) 'Valore dell''integrale: ', In
*
* Fine del programma
*
      stop
      end
```

Esercizio: scrivere un programma Fortran per calcolare il fattoriale di un intero n.

Algoritmo: $I_n = 1 + \sum_{k=1}^{n-1}(-1)^k n(n-1)\cdots(n-k+1) + (-1)^n n! I_0 = f(I_0)$

Nei calcoli non abbiamo usato il valore **esatto** $I_0^* = 0.63212055882856...$ ma il valore **arrotondato** $I_0 = 0.63212055882856$.

Come si **propaga** nel calcolo di I_n l'**errore di arrotondamento** sul dato di input $\epsilon_0 = I_0^* - I_0$?

Errore: $\epsilon_n = I_n^* - I_n = f(I_0^*) - f(I_0) = \underbrace{(-1)^n n!}_{\text{Coeff. di amplificazione}}\epsilon_0$

$\Rightarrow$ L'**algoritmo non è stabile**

Un nuovo algoritmo

Modifichiamo l'algoritmo nel modo seguente:

$$\begin{cases} I_n = 1 - nI_{n-1} \quad \Rightarrow \quad I_{n-1} = \dfrac{1 - I_n}{n} \\ I_n \to 0 \quad \text{per} \quad n \to \infty \ \textbf{(comportamento corretto)} \end{cases}$$

Algoritmo: $I_N = 0$, $I_{k-1} = \dfrac{1 - I_k}{k}$, $k = N, N-1, \ldots$

Come si **propaga** l'**errore di arrotondamento** sul dato di input $\epsilon_N = I_N^* - I_N = I_N^*$?

$$\epsilon_{N-1} = \frac{1-I_N^*}{N} - \frac{1-I_N}{N} = -\frac{\epsilon_N}{N}$$
$$\epsilon_{N-2} = \frac{1-I_{N-1}^*}{N-1} - \frac{1-I_{N-1}}{N-1} = \frac{\epsilon_N}{N(N-1)} \qquad \ldots$$

A ogni passo l'errore iniziale viene ridotto

$\Rightarrow$ l'**algoritmo** è **stabile**

$$I_{30} = 0 \longrightarrow I_{25}^{(30)} = \underline{0.03708621}625288$$
$$I_{35} = 0 \longrightarrow I_{25}^{(35)} = \underline{0.03708621}442374$$

$$I_{30} = 0 \longrightarrow I_{26}^{(30)} = \underline{0.03575837}742504$$
$$I_{35} = 0 \longrightarrow I_{26}^{(35)} = \underline{0.03575842}498278$$

Nota: Si può **stimare** l'**errore di arrotondamento** sul dato di **output** tramite la differenza tra due approssimazioni successive:

$$\epsilon_{25} \simeq I_{25}^{(35)} - I_{25}^{(30)} = -1.83e-009$$
$$\epsilon_{26} \simeq I_{26}^{(35)} - I_{26}^{(30)} = 4.76e-008$$

Condizionamento di un problema

Consideriamo il **problema** del calcolo di una funzione di una variabile reale f in un generico punto $x \in \mathbb{R}$: $\boxed{y = f(x)}$.

$$x \longrightarrow \boxed{f} \longrightarrow y$$

Vogliamo **misurare** quale effetto produce nel calcolo di y una **perturbazione** $\Delta x = x^* - x$ del dato di input.

Sviluppo in serie di Taylor:

$$\Delta y = y^* - y = f(x^*) - f(x) = f'(x)\Delta x + ...$$

Errore relativo:

$$\left|\frac{\Delta y}{y}\right| \leq \left|\frac{f'(x)}{f(x)}\right| |\Delta x| = \underbrace{\left|\frac{f'(x)x}{f(x)}\right|}_{\searrow} \left|\frac{\Delta x}{x}\right|$$

Numero di condizionamento del problema:

$$C_P := \left|\frac{f'(x)x}{f(x)}\right|$$

Se C_P è *"grande"* il problema è **malcondizionato**, cioè a **piccole perturbazioni** dei dati di input corrispondono **grandi variazioni** dei risultati. Se C_P è *"piccolo"* il problema è **ben condizionato**.

Osservazioni sul condizionamento

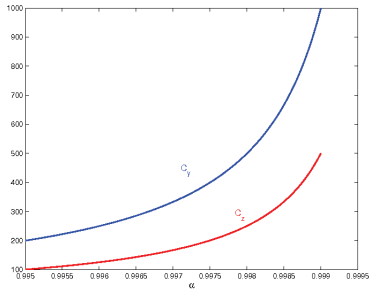
- Il **condizionamento non dipende** dall'algoritmo né dagli errori di arrotondamento.
- Il **condizionamento dipende** dal **problema** e dai **dati di input**: uno stesso problema può essere **ben condizionato** per alcuni valori dei dati, ma **mal condizionato** per altri valori.

Condizionamento: esempi

La soluzione del **sistema lineare** $\begin{cases} y + \alpha z = 1 \\ \alpha y + z = 0 \end{cases}$

è data da $y = \frac{1}{1-\alpha^2} = f(\alpha)$, $z = \frac{-\alpha}{1-\alpha^2} = g(\alpha)$.
$(\alpha^2 \neq 1)$

$$C_y = \left|\frac{f'(\alpha)\alpha}{y}\right| = \left|\frac{2\alpha^2}{1-\alpha^2}\right|$$
$$C_z = \left|\frac{g'(\alpha)\alpha}{z}\right| = \left|\frac{1+\alpha^2}{1-\alpha^2}\right|$$



$$\alpha = 0.5555 \rightarrow \begin{cases} y = 1.446299444 \\ z = -0.803419341 \end{cases} \qquad C_y = 0.89$$

$$\alpha = 0.5554 \rightarrow \begin{cases} y = 1.446067105 \\ z = -0.803145670 \end{cases}$$

$$\alpha = 0.9998 \rightarrow \begin{cases} y = 2500.250025 \\ z = -2499.749975 \end{cases}$$

$$\alpha = 0.9999 \rightarrow \begin{cases} y = 5000.250013 \\ z = -4999.749987 \end{cases} \qquad C_y = 5000$$

Esercizio.

Graficare (con gnuplot o con Matlab) C_y e C_z in funzione di α in diversi intervalli. Cosa succede se l'intervallo contiene il valore $\alpha = 1$?

Riferimenti bibliografici

L. Gori, *Calcolo Numerico*:
Cap. 1, Par. 1.1, 1.3 (fino errore relativo), Esempio 1.4.2, 1.5 (escluso caso bidimensionale e condizionamento del calcolo di una radice), 1.6 (concetto di stabilità ed esempio 1.6.1)

Per consultazione:
A. Quarteroni, F. Saleri, *Calcolo scientifico*, Springer, 2008

Programma di ricerca Orizzonte 2020

http://ec.europa.eu/research/horizon2020