

Proposta esercizi 1

Calcolo Numerico (LT Informatica), a.a. 2019-2020

*Dipartimento di Scienze Matematiche Fisiche e Informatiche
Università degli Studi di Parma*

Docente:
Prof.ssa Chiara Guardasoni

Tutor:
Ariel S. Boiardi *

1 Numeri macchina e calcolo in virgola mobile

Rappresentazione dei numeri reali in diverse basi Quando scriviamo un numero intero positivo N con la usuale *notazione posizionale decimale*, ad esempio $N = 1234567890$, sottintendiamo la sviluppo in potenze di 10:

$$1234567890 = 1 \cdot 10^9 + 2 \cdot 10^8 + 3 \cdot 10^7 + 4 \cdot 10^6 + 5 \cdot 10^5 + 6 \cdot 10^4 + 7 \cdot 10^3 + 8 \cdot 10^2 + 9 \cdot 10^1 + 0 \cdot 10^0,$$

dove, naturalmente, l'ordine degli addendi è ininfluente. In generale la scrittura posizionale decimale di un numero $N = d_n d_{n-1} \dots d_1 d_0$ sottintende lo sviluppo

$$N = d_n \cdot 10^n + d_{n-1} \cdot 10^{n-1} + \dots + d_1 \cdot 10 + d_0,$$

dove d_n, \dots, d_0 sono cifre fra 0 e 9.

Naturalmente non vi è ragione intrinseca per la scelta della base 10 rispetto a qualunque altro numero: altre civiltà hanno usato numerazioni in basi diverse e, ancora oggi, contiamo ore e minuti in base 60, ricorrendo alla rappresentazione decimale solo per i sottomultipli inferiori all'unità del secondo (anche se in effetti non capita spesso di usare frazioni dell'ora diverse da $\frac{1}{2}$ o $\frac{1}{4}$...).

Nel calcolatore, come ben noto, le cifre vengono rappresentate mediante stati fisici, ed è quindi conveniente per ragioni tecniche utilizzare una numerazione in base 2. Tale sistema di numerazione è detto binario e le cifre si riducono dai due simboli 0 e 1.

La possibilità di rappresentare un numero qualunque in una base a piacere è garantita dal seguente fondamentale risultato

Teorema 1. *Dato un intero $\beta > 1$, un numero reale $x \neq 0$ può essere univocamente espresso nella seguente forma*

$$x = \underbrace{\text{sign}(x)}_S \underbrace{(d_1 \beta^{-1} + d_2 \beta^{-2} + \dots)}_M \underbrace{\beta^p}_E \quad (1)$$

*Per dubbi e segnalazioni: arielsurya.boiardi@studenti.unipr.it

con $0 \leq d_i \leq \beta - 1$ per ogni i , $d_1 \neq 0$ e $d_i \neq \beta - 1$ per qualche i . Nell'equazione precedente, S è il segno del numero, M è detta mantissa, P parte esponente, l'esponente p è chiamato caratteristica.

Osserviamo che nel risultato precedente, per poter rappresentare un numero reale qualunque potremmo aver bisogno di infinite cifre, mentre nel calcolatore ogni variabile può, per ovvie ragioni, usare un numero finito di cifre. Se il numero di cifre significative è fissato a t , l'insieme dei numeri macchina, cioè quelli rappresentabili esattamente con t cifre in base β è

$$\mathbb{F}(\beta, t) = \{0\} \cup \left\{ x = \text{sign}(x) \beta^p \sum_{i=1}^t d_i \beta^{-i} \right\}.$$

La maggior parte dei calcolatori permettono di lavorare con variabili numeriche di lunghezze diverse, cui corrispondono diversi livelli di precisione e utilizzi di memoria.

Calcolo in virgola mobile Fino al 1985 i principali costruttori di calcolatori elettronici implementavano i numeri in virgola mobile ognuno con un proprio formato numeri e proprie convenzioni di calcolo. La necessità di scambiare dati fra diversi calcolatori portò nel 1985 all'istituzione dello standard IEEE 754[1] per i numeri in virgola mobile, oggi seguito da virtualmente tutti i produttori di sistemi di calcolo¹.

In MATLAB, ove non diversamente specificato, i numeri vengono rappresentati nel formato *double precision*², che utilizza 64 bits di memoria di cui 1 per il segno, 52 per la mantissa e 11 per la caratteristica. L'esponente è un intero da 11 bit, compreso quindi fra 0 e $2^{11}-1 = 2^{2047}$; per riuscire ad esprimere anche esponenti negativi è stato fissato $2^{10} - 1 = 1023$ come zero per l'esponente, pertanto $p \in [-1022, 1023]$ e i due esponenti 1024 e -1023 sono utilizzati valori speciali che rappresentano 0, ∞ o risultati di operazioni impossibili.

Esercizio 1: Alla luce della spiegazione sopra sulla rappresentazione in doppia precisione, come possiamo esprimere esplicitamente le quantità `realmin`, `realmax` ed `eps` restituite dalle omonime functions in MATLAB. Si vede bene che `realmin` differisce (parecchio!) dall'`eps` macchina, quale è la differenza fra questi due valori, quale è il più rilevante per misurare l'accuratezza di un calcolo?

Ora che abbiamo chiaro il significato di `realmin`, `realmax`, `eps` possiamo usarli per osservare alcuni comuni problemi

Esercizio 2: Che risultato darà il seguente calcolo

```
>> realmax + 1 - realmax
```

e perché? Avremo qualche differenza invertendo l'ordine? Perché?

¹Linguaggi, compilatori, interpreti, processori...

²Definito `binary64` nelle ultime revisioni dello standard

```
>> realmax - realmax +1
```

Come noto l'*epsilon macchina* `eps` è il più piccolo numero macchina positivo tale che $1 + \text{eps} > 1$ e pertanto

```
>> 1 + eps - 1

ans =

    2.220446049250313e-16

>> ans == eps

ans =

    logical

    1
```

possiamo coerentemente vedere che $1 + \text{eps}$ differisce da 1 proprio di `eps`. Sarà anche vero che $2 + \text{eps} > 2$? Perché?

I fenomeni osservati nel precedente esercizio sono noti come *overflow*, quando qualcosa viene trascurato perché supera il massimo valore rappresentabile, e *underflow*, quando qualcosa viene trascurato perché più piccolo della “sensibilità” del calcolatore.

Gli esempi appena mostrati mettono in evidenza come in aritmetica di macchina possano smettere di essere valide molte delle proprietà dell'aritmetica in campo reale. Va notato che alle volte sottovalutare questi problemi risulta catastrofico...³

Esercizio 3: Fissati $a = 0.233\,712\,58 \times 10^{-4}$, $b = 0.336\,784\,29 \times 10^2$, $c = -0.336\,778\,11 \times 10^2$, cosa ci aspettiamo dal confronto fra $(a + b) + c$ e $a + (b + c)$? E se il calcolo viene svolto in aritmetica di macchina? Come ti spiega il risultato?

Il fenomeno osservato è chiamato *eliminazione* e avviene quando vengono sottratti numeri molto simili fra loro. Naturalmente una eliminazione è presente in entrambi i calcoli dell'esercizio, ma data la differenza fra gli ordini di grandezza, nel primo calcolo è molto peggiore che nel secondo.

Esercizio 4: Calcolare in MATLAB la seguente somma

$$\sum_{k=1}^{10^4} \frac{1}{k},$$

confrontare il risultato ottenuto eseguendo la somma in verso naturale rispetto a quello ottenuto sommando da 10^4 fino a 1. Quale dei due risultati ci aspettiamo sia più affidabile? Usando la sintassi di MATLAB per le operazioni fra array il calcolo può essere svolto in modo molto naturale in un solo comando...come?

³<http://www-users.math.umn.edu/~arnold/disasters/>

Calcolo simbolico Avendo brevemente esplorato le difficoltà pratiche e teoriche del calcolo numerico, viene da chiedersi se non ci sia qualche modo per evitare i problemi visti, cioè riuscire a fare calcoli in modo automatico, ma senza dover fare troncamenti, rischiare overflow, underflow, eliminazioni... In effetti esistono diversi sistemi di calcolo automatico *simbolico*, cioè che implementano l'algebra del campo reale. Chi fosse curioso può sperimentare con uno di questi sistemi disponibile nella libreria `sympy` per Python

```
In [1]: from sympy import *  
  
In [2]: sqrt(2)**2 - 2  
  
Out[2]: 0
```

Questi sistemi però, per quanto interessanti, non sostituiscono il calcolo numerico.

2 Successioni, serie, convergenza

Approssimazioni di π Uno dei più antichi metodi per l'approssimazione di π è il metodo di esaustione formulato da Archimede di Siracusa. L'idea è di approssimare il perimetro di una circonferenza con il perimetro di un poligono ad essa inscritto, aumentando indefinitamente il numero di lati, la successione di poligoni esaurisce la circonferenza e permette di calcolare π .

Possiamo formulare l'algoritmo usando un linguaggio molto più moderno (e barando, visto che usiamo funzioni trigonometriche che richiedono di conoscere il valore di π) come segue: data una circonferenza unitaria un poligono di n lati ad essa inscritto ha i vertici a distanza $\frac{2\pi}{n}$ e lati di lunghezza $l = \sin\left(\frac{\pi}{n}\right)$. Il perimetro di tale poligono è

$$p_n = n \sin\left(\frac{\pi}{n}\right). \quad (2)$$

Ricordando che $\frac{\sin x}{x} \rightarrow 1$ per $x \rightarrow 0$ ricaviamo che

$$\lim_{n \rightarrow \infty} p_n = \pi.$$

Esercizio 5: Calcolare in MATLAB la successione dei perimetri utilizzando la formula in (2) e confrontare i risultati con il valore di `pi` restituito da MATLAB.

Un'altra famosa formula per l'approssimazione di π , o meglio di $\pi/4$, è quella di Leibniz

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

Esercizio 6: Implementare in MATLAB il calcolo di $\frac{\pi}{4}$ mediante formula di Leibniz, confrontare i risultati arrestando le somme a diversi livelli e con il valore di `0.25*pi`.

Pensiamo ora ad un cerchio di raggio unitario, la sua area sarà π , mentre l'area del quadrato ad esso circoscritto sarà 1. Se riuscissimo a misurare, anche in modo

approssimato, il rapporto fra l'area del cerchio quella del rettangolo avremmo una approssimazione per $\frac{\pi}{4}$. Se immaginiamo di lanciare una freccetta a caso sul quadrato circoscritto, la probabilità che questa cada nel cerchio è pari al rapporto fra le due superfici. Naturalmente un solo lancio non dice molto, ma eseguendo tanti lanci il rapporto fra il numero di freccette cadute nel cerchio e quello cadute fuori fornisce una approssimazione sempre più precisa di $\frac{\pi}{4}$.

Esercizio 7: Implementare in MATLAB l'approssimazione di π mediante il lancio delle freccette descritta sopra.

Soluzione: Una implementazione semplice è fornita nello script di seguito

```
clear; close all; clc

Nf = 1000; % Numero lanci
Nin = 0; % Numero freccette nel cerchio
pi_est = zeros(Nf,1);

for k=1:Nf
    % La posizione casuale della freccetta ha coordinate x e y
    x = -1+2*rand;
    y = -1+2*rand;

    % Contiamo quante freccette sono cadute all'interno del cerchio
    Nin = Nin + sum(norm([x,y], 2) <= 1);

    % Per stimare pi
    pi_est(k) = Nin / k * 4;
end
fprintf("Ultima stima %f", pi_est(end))
plot(pi_est)
```

Riferimenti bibliografici

- [1] «IEEE Standard for Binary Floating-Point Arithmetic». In: *ANSI/IEEE Std 754-1985* (1985), pp. 1–20. DOI: 10.1109/IEEESTD.1985.82928.

⁴L'approssimazione dell'area come somma di tentativi casuali qui descritta è la base dei metodi Monte Carlo.