

Prologo

Il Patriot e lo Scud

Durante la guerra del Golfo, il 25 Febbraio del 1991, un missile americano da difesa Patriot lasciò *passare* un missile Scud nemico.

Il missile Scud colpì un accampamento militare uccidendo 28 soldati.

Il problema fu individuato nel calcolo della differenza tra due numeri in virgola mobile ottenuti convertendo e scalando un numero intero di un registro di temporizzazione.



Missile Patriot



Missile Scud

Prologo

Il breve volo del razzo vettore Ariane 5

Il 4 Giugno 1996 venne lanciato il primo razzo Ariane 5. Tutto andò bene per 36 secondi. Poi improvvisamente il razzo deviò dalla sua traiettoria e si autodistrusse.

Il problema risiedeva nel Sistema di Riferimento Inerziale che generò un "operation exception" convertendo un numero in doppia precisione a 64 bit in un intero a 12 bit. Ciò produsse l'invio di un messaggio errato di diagnostica al computer di bordo che lo interpretò come un dato relativo alle condizioni di volo. **FINE.**

Ironicamente l'errore di calcolo proveniva da una sezione di codice ereditata da Ariane 4 i cui risultati non erano necessari dopo il decollo.



Ariane 5

Prologo

Il Vancouver Stock Exchange

Verso il 1982, la borsa di Vancouver introdusse un nuovo indice inizializzandolo al valore di: 1,000.000 (mille con tre cifre decimali).

L'indice veniva aggiornato dopo ogni transizione condotta sulla piazza di Vancouver. L'indice era basato sulla quotazione di circa 1,400 titoli cosicché in media il valore dell'indice veniva ricalcolato circa 3,000 volte al giorno sempre con una precisione di tre cifre decimali.

Dopo 22 mesi di attività, l'indice raggiunse quota: 520. La causa di tutto ciò non era dovuta ad un periodo di recessione economica, bensì al fatto che ogni nuovo valore calcolato veniva troncato alla terza cifra decimale anziché essere arrotondato.

Ad esempio il valore effettivo: 540.32567 veniva troncato a 540.325 anziché essere arrotondato a 540.326. Così facendo l'indice perdeva in media 1 punto al giorno quindi circa 20 punti al mese.

La sostituzione dell'operazione di troncamento con quella di arrotondamento portò direttamente l'indice ad assumere il valore corretto: 1,098.892.



Prologo

Elezioni del parlamento tedesco

Il regolamento relativo alle elezioni del parlamento tedesco prevede che un partito con meno del 5% dei voti non abbia diritto ad alcun rappresentante.

Il partito dei Verdi nelle elezioni del 1992 sembrò che avesse raggiunto esattamente la quota del 5% prima che si scoprisse (una volta che i risultati erano già stati annunciati) che aveva raccolto per l'esattezza il 4.97% dei voti.

La percentuale effettiva di voto era stata arrotondata a 5.0%



Parlamento tedesco

Basi di numeri

I processori presenti nei calcolatori elettronici non operano in base 10 bensì in base 2 a causa della semplice rappresentabilità dello **stato 1/0** (on/off, acceso/spento, passa/non passa, si/no) da parte di un **transistor**.

Il transistor è l'unità elementare che costituisce la struttura fisica di un processore.



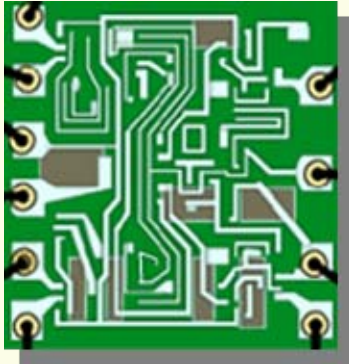
STRUTTURA di un TRANSISTOR

The transistor is a three terminal, solid state electronic device. In a three terminal device we can control electric current or voltage between two of the terminals by applying an electric current or voltage to the third terminal. This three terminal character of the transistor is what allows us to make an amplifier for electrical signals, like the one in our radio. With the three-terminal transistor we can also make an electric switch, which can be controlled by another electrical switch. By cascading these switches (switches that control switches that control switches, etc.) we can build up very complicated logic circuits.

These logic circuits can be built very compact on a silicon chip with 1,000,000 transistors per square centimeter. We can turn them on and off very rapidly by switching every 0.000000001 seconds. Such logic chips are at the heart of your personal computer and many other gadgets you use today.

Basi di numeri

I CIRCUITI INTEGRATI



In 1958 and 1959, Jack Kilby at Texas Instruments and Robert Noyce at Fairchild Camera, came up with a solution to the problem of large numbers of components, and the integrated circuit was developed. Instead of making transistors one-by-one, several transistors could be made at the same time, on the same piece of semiconductor. Not only transistors, but other electric components such as resistors, capacitors and diodes could be made by the same process with the same materials.

For more than 30 years, since the 1960's, the number of transistors per unit area has been doubling every 1.5 years. This fantastic progression of circuit fabrication is known as Moore's law, after Gordon Moore, one of the early integrated circuit pioneers and founders of Intel Corp. The Nobel Prize in Physics 2000 was awarded to Jack Kilby for the invention of the integrated circuit.

Occorre perciò rammentare che i calcolatori lavorano in base 2 con un numero finito di bit, tipicamente 32 bit.

BInary **digiT**

Il **bit** è l'unità elementare numerica con cui il processore lavora.

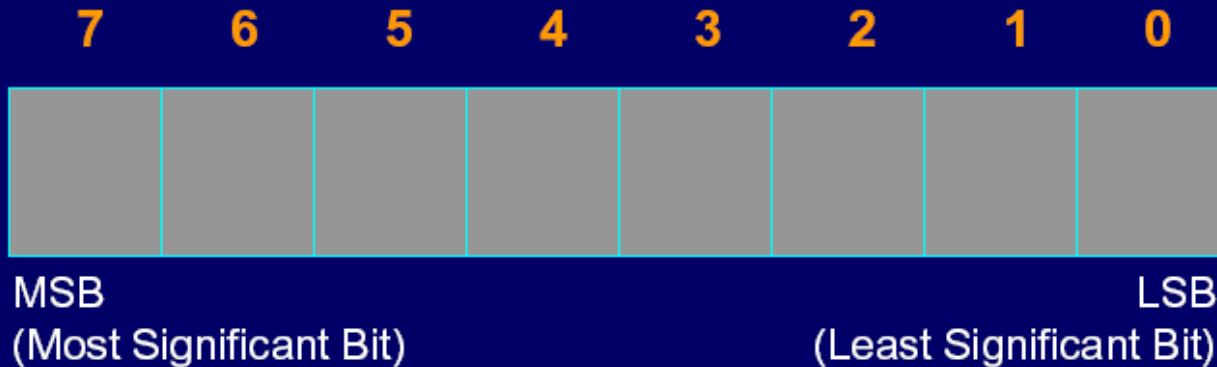
Il bit può assumere solo due stati: 1 o 0 (passa corrente, non passa corrente).

Basi di numeri

I bit *“non stanno mai da soli”* in un processore, bensì sono raccolti in gruppi di 8 bit e formano 1 byte.

- 1 byte \equiv 8 bit
- Il bit viene abbreviato con b (1 Mb \equiv 1,000,000 bit)
- Il byte viene abbreviato con B (1 MB \equiv 1,000,000 byte)

• Insieme ordinato di 8 bit



$2^8 = 256$ combinazioni possibili (valori rappresentabili)

Basi di numeri

La memoria centrale del computer (cache e RAM) o quella periferica (HD, CD-ROM, DVD, unità nastro, ...) è misurata in kB, MB, GB, TB, PB.

- 1 kB \equiv 1,000 byte
- 1 MB = 1,000,000 byte
- 1 GB \equiv 1,000 MB \equiv 1,000,000,000 byte
- 1 TB \equiv 1,000 GB \equiv 1,000,000 MB \equiv 1,000,000,000,000 byte
- 1 PB \equiv 1,000 TB \equiv 1,000,000 GB \equiv 1,000,000,000,000,000 byte

N.B.: esiste in realtà l'ambiguità tra KB \equiv 1024 byte e kB \equiv 1000 byte (conseguentemente 1 MB può rappresentare sia 1,000,000 byte sia 1,048,576 = 1024 \times 1024 byte).



Basi di numeri

Per avere un'idea delle dimensioni tipiche delle memorie fisiche citate in precedenza:

- Cache processori AMD e INTEL: 512 – 2048 KB
- Floppy: 1.44 MB
- CD-ROM: 650 – 700 MB
- DVD: 4.7 GB
- HD: 80 — 500 GB
- Unità backup a nastro ad altissima densità: 2.1 TB



Basi di numeri

1. Tutto il testo della Bibbia è contenuto in circa 7 MB.
2. Il cervello umano si stima possa contenere circa 50 GB di dati (memoria).
3. In 100 GB è possibile memorizzare circa 300,000 romanzi o 150 ore di televisione.
4. Un'azienda di telecomunicazioni con circa 20 milioni di utenti ha una banca dati clienti dell'ordine del TB e la disponibilità di caselle postali richiede un'occupazione di qualche PB.
5. Il film "Il signore degli anelli" è giunto ad occupare 27 TB durante l'elaborazione software degli effetti speciali ed il montaggio delle scene.
6. Contenuto di Internet: 2 PB (1 PB \equiv 1,000 TB \equiv 1,000,000 GB) è anche il contenuto della banca dati di Google (www.google.com)
7. Gli studi sul genoma umano coinvolgenti 20 centri di ricerca specializzati e 100 università producono ogni anno circa 7 PB di dati.



Basi di numeri

Come detto un byte \equiv 8 bit. Un esempio di rappresentazione in bit di un byte è:
10011010 oppure 00001111

8 bit possono identificare (discriminare) $2^8 = 256$ distinte combinazioni di numeri o di caratteri.

- Un byte se presentato come carattere (**character**) è in grado di rappresentare 256 caratteri distinti.
- Un byte se presentato come numero intero positivo (**unsigned integer**) descrive i numeri tra 0 e 255
- Un byte se presentato come numero intero con segno (**signed integer**) descrive i numeri tra -128 e +127



Interi binari

Un **intero binario** x è una **sequenza finita** di cifre 0 e 1: $x = (a_m a_{m-1} \dots a_2 a_1 a_0)_2$

Si noti che le cifre a_n possono assumere **soltanto** il valore: 0 o 1.

In base 10 è possibile rappresentare lo stesso numero x utilizzando la seguente formula:

$$x = a_m 2^m + a_{m-1} 2^{m-1} + \dots + a_2 2^2 + a_1 2^1 + a_0$$

Ad esempio: $x = (110101)_2 \equiv 2^0 + 2^2 + 2^4 + 2^5 = 1 + 4 + 16 + 32 = (53)_{10}$



Frazioni binarie

Una **frazione binaria** x è una **sequenza (teoricamente anche infinita)** di cifre 0 e 1:

$$x = (.a_1a_2 \dots a_m \dots)_2$$

Si noti che le cifre a_n possono assumere **soltanto** il valore: 0 o 1.

In base 10 è possibile rappresentare lo stesso numero x utilizzando la seguente formula:

$$x = a_1 2^{-1} + a_2 2^{-2} + \dots + a_m 2^{-m} + \dots$$

Ad esempio: $x = (.1101)_2 \equiv 2^{-1} + 2^{-2} + 2^{-4} = 0.5 + 0.25 + 0.0625 = (0.8125)_{10}$



Frazioni binarie

Si desidera sottolineare il concetto di **sequenza teorica infinita** necessaria per rappresentare **alcuni** numeri in notazione binaria.

$$\begin{aligned}\text{Ad esempio: } (.01010101010101\dots)_2 &= 2^{-2} + 2^{-4} + 2^{-6} + \dots = 2^{-2} (1 + 2^{-2} + 2^{-4} + \dots) = \\ &= 2^{-2} (1 + 4^{-1} + 4^{-2} + 4^{-3} + \dots) = 2^{-2} \frac{1}{1 - \frac{1}{4}} = \frac{1}{3}\end{aligned}$$

$$\text{Si noti che: } (.0101010101\dots)_2 = (.333333333\dots)_{10} = (.1)_3$$

Ovvero un numero frazionario **illimitato** in una base può invece essere rappresentato con un numero **finito** di cifre in un'altra base.

$$\begin{aligned}\text{Altri esempi: } (.110011001100\dots)_2 &= (.8)_{10} \\ \frac{1}{10} &= (.1)_{10} = (.0631\ 4631\ 4631\ 4631\dots)_8 = (.0001\ 1001\ 1001\ 1001\dots)_2\end{aligned}$$



Basi di numeri

Insegnamento: dato che il computer lavora con un **numero finito di cifre binarie NON è possibile** rappresentare in **base 2** in modo esatto:

- Qualsiasi numero irrazionale
- Qualsiasi numero razionale illimitato in base 10 (come ad esempio: $1/3$)
- Alcuni numeri razionali finiti in base 10 (come ad esempio: 0.8)

Non esiste corrispondenza biunivoca tra numeri reali e numeri rappresentati dal computer (*floating point*).

Il motivo della mancanza di corrispondenza è che il numero di bit, con cui un numero viene rappresentato su computer, **è finito**.



Basi di numeri

Tutti i processori presenti nei personal computer e nelle workstation di ultima generazione sono a 32 bit. Sono cioè in grado di indirizzare 4 GB di RAM.

AMD (con i processori della famiglia Opteron™ e Athlon 64™) ed Intel (con i processori della famiglia Itanium™) stanno introducendo, con una certa difficoltà, i primi processori a 64 bit. Questi ultimi sono in grado di indirizzare 18,446 PB di RAM equivalenti a 18 EB (Exabyte) di RAM.

I numeri interi e quelli decimali (in virgola mobile, *floating point*) sono rappresentati utilizzando usualmente 32 o 64 bit di memoria.

- Con 32 bit si possono rappresentare 4,294,967,296 (4 miliardi, 4 giga) di numeri differenti.
- Con 64 bit si possono rappresentare 18,446,744,073,709,551,616 (18 miliardi di miliardi, 18 exa) di numeri differenti.



Rappresentazione degli Interi

Iniziamo focalizzando l'attenzione sulla rappresentazione dei numeri tramite 32 bit (4 byte).

I **numeri interi** rappresentabili con 32 bit sono 2^{32} :

- da 0 a 4,294,967,295 (**unsigned integer**)
- da -2,147,483,648 a +2,147,483,647 (**signed integer**)

N.B.: Nel caso di interi con segno i due estremi non sono simmetrici in quanto occorre tenere conto anche dell'elemento nullo, ovvero lo zero: 0.



Rappresentazione degli Interi

Ciò sta a significare che lavorando con un **unsigned integer** a 32 bit se eseguiamo la seguente operazione:

$$4,294,967,295 + 1$$

il computer va in errore (**overflow**) in quanto NON è in grado di rappresentare un numero di tale grandezza. In realtà l'errore di overflow tra interi viene gestito come eccezione da parte del processore in modo del tutto "particolare" come se gli estremi dei numeri interi (0 e $2^{32}-1$) si toccassero.

Quindi: $4,294,967,295 + 1 = 0$ se si lavora con numeri interi senza segno (**unsigned integer**).

Equivalentemente: $+2,147,483,647 + 1 = -2,147,483,648$ se si lavora con numeri interi con segno (**signed integer**).



Rappresentazione dei Reali

Nel caso di rappresentazione **binaria** di numeri reali con architettura del processore basata sullo standard *floating point* dell'associazione IEEE (Institute of Electrical and Electronics Engineers) si ha:

$$fl(x) = \sigma \cdot (1.a_1a_2 \dots a_p)_2 \cdot 2^e$$

Con $fl(x)$ si indica la rappresentazione (approssimazione) da parte del computer del numero reale x .

σ rappresenta il **segno** di x : + o - (a livello di bit 1 o 0).

P è il numero di cifre binarie (0 o 1) della **mantissa** e dipende dal tipo di rappresentazione in virgola mobile: 23 in singola precisione, 52 in doppia precisione

e è il numero di cifre binarie (0 o 1) dell'**esponente** e dipende dal tipo di rappresentazione in virgola mobile: 8 in singola precisione, 11 in doppia precisione. Si ha che: $L \leq e \leq U$



Rappresentazione dei Reali

È quindi possibile riassumere nella tabella sottostante i dati relativi alla rappresentazione dei numeri reali nel sistema *floating point* IEEE riconosciuto internazionalmente e su cui i costruttori di processori: AMD, HP, IBM, INTEL, MOTOROLA, SUN si sono allineati.

Specificatamente si ha un formato dei numeri in virgola mobile in **singola precisione** (ogni numero occupa 4 byte) ed uno in **doppia precisione** (ogni numero occupa 8 byte).

	Single	Double
Sign width in bits	1	1
P	23	52
E _{max}	127	1023
E _{min}	-126	-1022
Exponent width in bits	8	11
Format width in bits	32	64



Rappresentazione dei Reali

Dalla rappresentazione in formato *floating point* dei numeri reali risulta che:

Il numero più piccolo positivo in formato IEEE è: $tiny = 2^{L-1}$

Il numero più grande positivo in formato IEEE è: $huge = (1 - 2^{-P})2^U$

Il macheps vale:

$$macheps = 2^{-P}$$

Analogo discorso, simmetrizzando, vale per i numeri negativi.

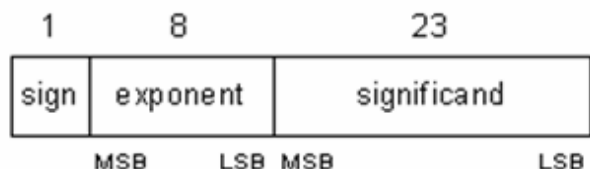
N.B.: dalla tabella di pagina precedente si ha: $L = E_{min}$ $U = E_{max}$

Vedi anche note aggiuntive alla fine del presente capitolo.

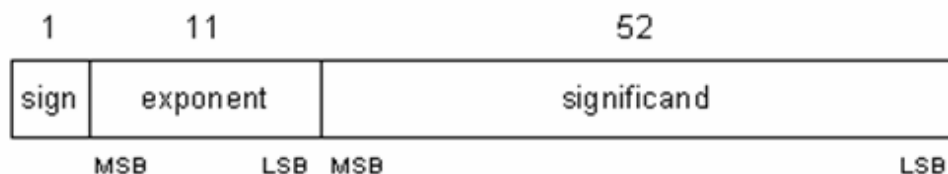


Rappresentazione dei Reali

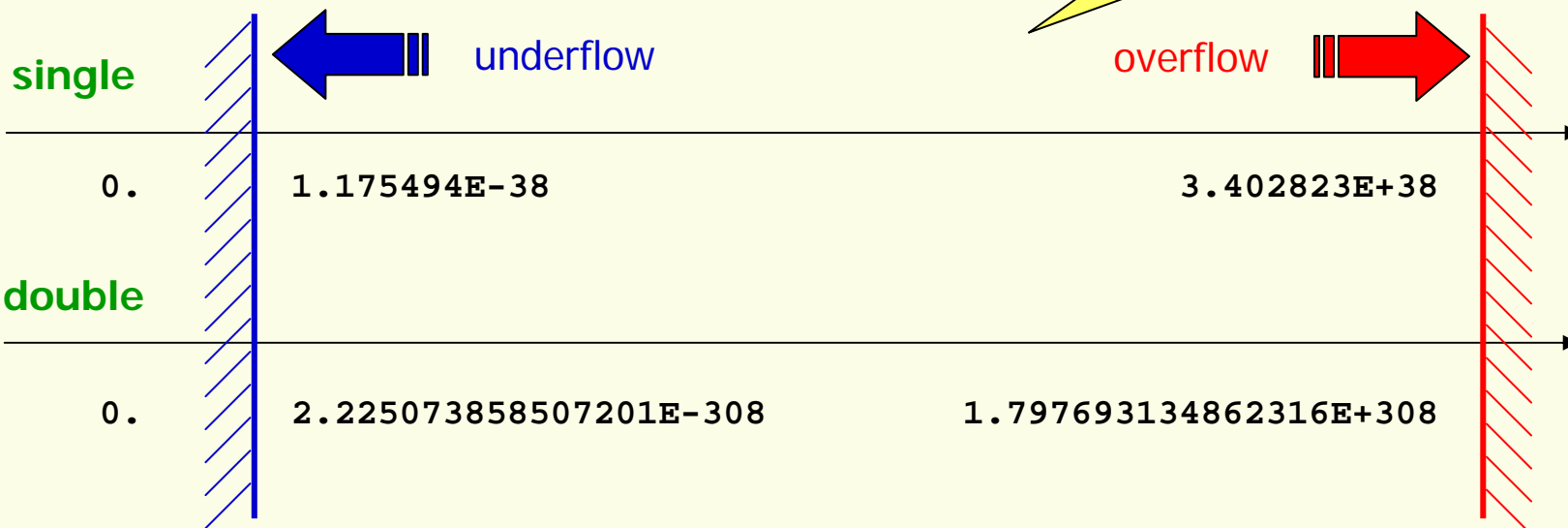
Logical Structure of the IEEE Single and Double Formats



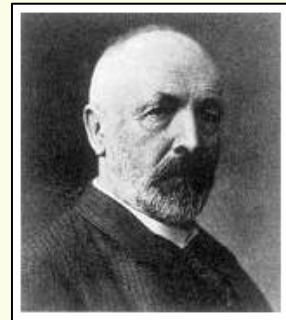
Double Format



Lo stesso schema è valido (simmetrizzando il diagramma) anche per la porzione di numeri negativi



Rappresentazione dei Reali



George Cantor
1845-1918

Il numero significativo di cifre in **singola precisione** risulta essere: 7

Il numero significativo di cifre in **doppia precisione** risulta essere: 15

Il concetto di continuo reale (numeri razionali + numeri irrazionali) viene così a perdersi.

L'insieme finito dei numeri in *floating point* è un sottoinsieme dell'insieme infinito dei razionali che viene definito anche come Polvere di Cantor.

Per avere un esempio, lavorando in singola precisione, la quantità più piccola positiva che può essere aggiunta a 3 per ottenere tramite computer un numero diverso da 3 è: **3.576279E-07**.

A livello di granularità quindi 3 e **3.0000003576279** sono due numeri "adiacenti" nella rappresentazione in singola precisione da parte del computer. Gli infiniti numeri reali appartenenti a tale intervallo **NON** sono rappresentabili da parte del computer.

Conseguentemente in singola precisione: $3. + 1.07E-07 = 3.$



Rappresentazione dei Reali

In base alle considerazioni finora condotte, scaturisce che: $fl(x) = x(1 + \varepsilon)$

dove: $-2^{-P} \leq \varepsilon \leq 2^{-P}$

Si desidera sottolineare che la distanza tra due numeri **adiacenti** nel sistema *floating point* cresce con il valore assoluto di tali numeri.



Arrotondamento e Troncamento

$$fl(x) = \sigma \cdot (1.a_1a_2 \dots a_p)_2 \cdot 2^e$$

Definizione: il numero P di cifre binarie della mantissa è detto: *precisione*.

Nota bene che P misura la precisione in base 2.

In genere la precisione in base 10, corrispondente a quella P in base 2, viene indicata con t .

Data la impossibilità di rappresentare correttamente un numero reale x tramite sistema in virgola mobile $fl(x)$, è necessario operare una delle seguenti scelte :

- **Arrotondamento** (rounding): viene scelto il numero $fl(x)$ che si discosta meno da x
- **Troncamento** (chopping): viene scelto il numero con mantissa minore



Arrotondamento e Troncamento

Esempio: dato il numero reale $x = 0.123456789$ la corrispondente rappresentazione $fl(x)$ in base 10 con precisione $t = 8$ è:

- con arrotondamento **0.12345679**
- con troncamento: **0.12345678**

L'arrotondamento è preferito al troncamento in quanto:

- dato un numero reale x per cui $x \neq fl(x)$ se lo si **tronca** risulta sempre:
 $fl(x) < x$
- viceversa, statisticamente, la rappresentazione $fl(x)$ di x tramite **arrotondamento** risulta essere per metà $fl(x) < x$ e per metà $fl(x) > x$.



Operazioni elementari in virgola mobile

Anche le operazioni elementari (+, -, *, /) debbono sottostare alle limitazioni di rappresentazione del sistema *floating point*.

Il risultato di un'operazione elementare tra due numeri in *floating point* **può** essere un numero che a sua volta deve essere arrotondato.

Si indichi con \odot la generica operazione elementare svolta senza arrotondamento e con \otimes la corrispondente operazione in virgola mobile eseguita con arrotondamento.

Si assume che: $x \otimes y \equiv fl(x \odot y) \equiv (x \odot y)(1 + \varepsilon)$

Quanto rappresentato simbolicamente sta a significare che il risultato dell'operazione in virgola mobile è equivalente al risultato dell'operazione elementare eseguita correttamente e successivamente rappresentato in virgola mobile.

A tal fine si rammenta che: $fl(x) = x(1 + \varepsilon)$



Misure dell'errore

Come più volte detto, il calcolatore non produce risultati esatti.

È opportuno allora introdurre delle misure quantitative degli errori commessi, al fine di poter analizzare criticamente gli esiti della risoluzione numerica di un problema.

Se x è il valore esatto di una grandezza e \hat{x} quello approssimato, si ha:

Definizione: errore assoluto $\Delta x = |x - \hat{x}|$

Definizione: errore relativo $\varepsilon_x = \frac{\Delta x}{|x|} = \left| \frac{x - \hat{x}}{x} \right|$

L'**errore assoluto** può essere anche molto elevato, ma ancora accettabile, se l'ordine di grandezza della variabile cui si riferisce è grande.

L'**errore relativo** è poco significativo quando l'ordine di grandezza della variabile cui si riferisce è molto piccolo (si è vicini al limite di precisione del calcolatore).



Sorgenti di errore

Verranno presentate nel seguito le possibili sorgenti di errore nella soluzione numerica di un modello reale.

Errore di modellazione

È l'errore compiuto nella descrizione matematica della porzione di realtà (fenomeno fisico) che si desidera modellare. Si considera errore l'inevitabile semplificazione che il modello matematico adotta rispetto alla realtà.

Esempio: modellazione della dinamica di un proiettile che si muove nell'atmosfera

$$m \frac{d^2 \mathbf{x}(t)}{dt^2} = -mg\mathbf{k} - b \frac{d\mathbf{x}}{dt}$$

Il termine di attrito $b \geq 0$ è positivo ma è costante in questo modello. In realtà è ragionevole pensare che sia funzione della velocità stessa del proiettile. Inoltre, non sono state considerate esplicitamente la densità e la viscosità dell'atmosfera che dipendono dalla temperatura della stessa,



Sorgenti di errore

Errore grossolano

Quando non c'era il computer questo era il tipico errore aritmetico di calcolo.

Con l'introduzione dei computer ci si è spostati sugli errori programmatici.

I più subdoli da trovare sono quelli che appartengono a grosse porzioni di codice e che non mandano in errore l'esecuzione ma che producono dei risultati accettabili. Esempio: $Nu \propto Re^{4/5} Pr^{1/3} \rightarrow Pr^{(1/3)}$ anziché $Pr^{(1./3.)}$

$Re^{(4/5)}$ anziché $Re^{(4./5.)}$

Alcuni suggerimenti:

Suddividere il programma in porzioni di codice più piccole (**functions**) che siano più facili da testare e che una volta convalidate possano essere riutilizzabili con sicurezza.

- Eseguire dei test del codice che forniscano un output già noto da confrontare.
- Mantenere un approccio scettico e conservativo nei confronti del codice analizzando con senso critico i risultati ottenuti.

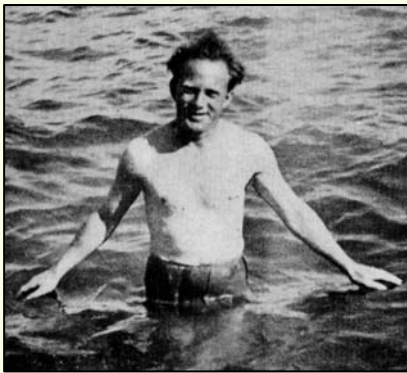


Sorgenti di errore

Errore di misura

Il raggio di un elettrone vale: $(2.81777 + \varepsilon) \times 10^{-13} \text{ cm}$, $|\varepsilon| \leq 0.00011$

Questo errore, legato alla precisione della misura, non può essere eliminato e coscientemente dobbiamo sapere e ricordare che condiziona tutti i calcoli relativi alla risoluzione di modelli atomici della materia.



Werner Heisenberg (1901-1976)

Sorgenti di errore

Errore di arrotondamento/troncamento

Questo è l'errore che produce i maggiori problemi nella risoluzione numerica; specificatamente nella risoluzione di sistemi lineari.



Sorgenti di errore

Errore di approssimazione

Lo si definisce anche **errore di discretizzazione** ed è una delle maggiori fonti di errore.

In genere questo errore ha origine nel momento in cui si sostituisce un problema non risolubile in termini computazionali con uno viceversa risolubile.

Ad esempio lo sviluppo in serie di Taylor: $e^x \approx 1 + x + \frac{x^2}{2}$ contiene un errore di approssimazione. Lo stesso può essere detto per:

$$\int_0^1 f(x) dx \approx \frac{1}{n} \sum_{j=1}^n f\left(\frac{j}{n}\right)$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$



Sorgenti di errore

Errore di perdita di cifre significative

È dovuto al numero limitato di cifre significative disponibili in un computer.

Esempio: si calcoli la funzione $f(x) = x(\sqrt{x+1} - \sqrt{x})$ su un computer con 6 cifre significative e con arrotondamento. Si ha:

x	f(x) calcolata	f(x) corretta	errore assoluto	errore relativo
1	0.414210	0.414214	4E-06	9.65684E-06
10	1.54340	1.54347	7E-05	4.53524E-05
100	4.99000	4.98756	0.00244	0.000489217
1,000	15.8000	15.8074	0.0074	0.000468135
10,000	50.0000	49.9988	0.0012	2.40006E-05
100,000	100.000	158.113	58.113	0.367540936

Insegnamento: quando due numeri sono quasi uguali (per la precisione del computer) e li si sottrae, si subisce una perdita di cifre significative del calcolo. In certi casi può essere difficile rilevare questo tipo di errore e anche quando lo si è identificato può essere complicato eliminarlo (spesso occorre riformulare il problema).



Sorgenti di errore

Errore di perdita di cifre significative

Ecco un esempio di riformulazione di un problema numerico:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + \dots$$

In questo caso, se $x > 0$, per calcolare e^{-x} si debbono sommare dei termini che alternativamente sono positivi e negativi e che divengono man mano più piccoli, cioè sempre più prossimi alla precisione del calcolatore.

È possibile migliorare notevolmente il problema di perdita di cifre significative modificando l'approccio e calcolando:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots \quad \Rightarrow \quad e^{-x} = \frac{1}{e^x}$$

Sorgenti di errore

Errore di underflow

Lavorando in singola precisione con processore in standard IEEE, il più piccolo numero positivo rappresentabile è $\approx 1.18\text{E}-38$.

La seguente operazione produce un errore di **underflow**:

$$x = 1.e-30 * 1.e-20$$

In genere il linguaggio con cui è stato scritto il programma pone uguale a zero tale operazione (in linea di principio illegale). Il flusso del programma prosegue senza alcun **Warning**.



Sorgenti di errore

Errore di overflow

Lavorando in singola precisione con processore in standard IEEE, il più grande numero positivo rappresentabile è $\approx 3.40\text{E}38$.

La seguente operazione produce un errore di **overflow**:

$$x = 1.e30 * 1.e20$$

Questo errore non è risolubile come nel caso dell'underflow. In genere il linguaggio utilizzato e le opzioni di compilazione con cui è stato prodotto l'eseguibile, bloccano il flusso del programma e resettano lo stato di errore del processore emettendo un **Fatal Error**. Il programma viene quindi abortito.

(continua)



Sorgenti di errore

Errore di overflow

Pericolo: alcuni linguaggi di programmazione e le corrispondenti opzioni di compilazione, quando si verifica un errore di overflow, **NON** emettono alcun messaggio e proseguono il calcolo assegnando alla variabile incriminata il valore: **Inf** (infinito). Se l'utente non analizza tempestivamente il valore di tale variabile **NON** può accorgersi dell'errore (il più delle volte dovuto a errore nei dati di input o a errori nella struttura del codice o dell'algoritmo o a errori concettuali).

Il problema maggiore è che se nel prosieguo del calcolo la variabile **Inf** viene moltiplicata ad esempio per 0 oppure si divide una variabile per **Inf** il risultato finale è: 0 e si perde definitivamente traccia di tale errore.

Consiglio: un errore di overflow deve bloccare l'esecuzione del programma. L'utente deve analizzare (debuggare) il codice in cerca della sorgente di errore e correggerla (bug fixing). Tale attività può richiedere molto tempo ma è necessaria.



Sorgenti di errore

Errore di propagazione

Qualsiasi errore nei dati del problema o generato durante l'esecuzione dell'algoritmo risolutivo viene propagato col procedere dei calcoli.

Esempio: si desidera calcolare un valore di $f(x)$. Il risultato in genere non è $f(x)$ bensì un'approssimazione: $\tilde{f}(x)$

Supponiamo di voler calcolare $f(x)$ in x_v (valore *vero*) ma che x_v sia approssimato nel computer da x_a (valore *approssimato*). Per questo motivo anziché calcolare $f(x_v)$ valuteremo $\tilde{f}(x_a)$. Risulterà pertanto:

$$f(x_v) - \tilde{f}(x_a) = \underbrace{[f(x_v) - f(x_a)]}_{\text{Errore di propagazione}} + \underbrace{[f(x_a) - \tilde{f}(x_a)]}_{\text{rumore}}$$

Errore di propagazione. Esso è dovuto all'errore di rappresentazione dei dati di input. Si noti l'ipotesi di calcolo della funzione $f(x)$ con precisione assoluta.

È il **rumore** nel calcolo in *floating point* della funzione $f(x)$.

Sorgenti di errore

Errore dovuto all'algoritmo

A seconda dell'algoritmo adottato per risolvere uno specifico problema numerico si avrà una certa sequenza delle operazioni elementari.

Cambiando algoritmo si modifica anche la sequenza delle operazioni. Conseguentemente l'errore di propagazione avrà un'evoluzione che è funzione dell'algoritmo adottato.

Esempio: calcolare il valore della sommatoria: $\sum_{i=1}^{1,000,000} \frac{1}{i}$

Algoritmo 1

```
somma = 0.  
for i = 1 to 1,000,000  
    somma = somma + 1./i  
next i
```

Algoritmo 2

```
somma = 0.  
for i = 1,000,000 to 1 step -1  
    somma = somma + 1./i  
next i
```

Propagazione dell'errore nelle operazioni elementari

Si desidera ora analizzare l'errore commesso durante l'uso delle comuni operazioni aritmetiche: (+, -, *, /). Siano:

⊙ operazione aritmetica esatta

⊗ operazione aritmetica svolta con la precisione finita del calcolatore

$x_a \approx x_v$ approssimazione x_a e valore esatto x_v

$y_a \approx y_v$ approssimazione y_a e valore esatto y_v

Si desidera calcolare $x_v \odot y_v$ ma si valuta in realtà $x_a \otimes y_a$. L'errore commesso è:

$$x_v \odot y_v - x_a \otimes y_a = [x_v \odot y_v - x_a \odot y_a] + \underbrace{[x_a \odot y_a - x_a \otimes y_a]}$$

il **secondo** termine tra parentesi quadre è l'**errore locale** introdotto dalla imprecisione dell'aritmetica del calcolatore. In genere si assume che:

$$x_a \otimes y_a = fl(x_a \odot y_a)$$

Ciò sta a significare che la quantità $x_a \odot y_a$ è calcolata con esattezza e quindi è arrotondata/troncata alla precisione del calcolatore.

$$x_a \otimes y_a = (x_a \odot y_a)(1 + \varepsilon) \quad |\varepsilon| < \text{macheps}$$



Propagazione dell'errore nelle operazioni elementari

Il **primo** termine tra parentesi quadre: $x_v \odot y_v - x_a \odot y_a$ è detto **errore di propagazione**.

Per comprendere il concetto di propagazione dell'errore si supponga di avere un legame tra due grandezze y e x : $y = f(x)$. Se si perturba di poco x risulta che:

$$\hat{y} = y + f'(x)(\hat{x} - x)$$

da cui è possibile determinare il legame tra l'errore relativo compiuto sulla variabile dipendente y rispetto quella indipendente x : $\varepsilon_y = \kappa(x)\varepsilon_x$ con:

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

Definizione: la funzione $\kappa(x)$, che lega l'errore relativo della variabile y a quello della variabile x , si chiama **numero di condizionamento**.

Il numero di condizionamento rappresenta il fattore di amplificazione dell'errore relativo. Questo concetto può essere esteso anche al caso di matrici e vettori.



Propagazione dell'errore nelle operazioni elementari

Per quanto riguarda le quattro operazioni elementari è possibile dimostrare che:

$$y = a \cdot x \quad \varepsilon_y = \left| \frac{ax}{ax} \right| \varepsilon_x = \varepsilon_x$$

$$y = \frac{a}{x} \quad \varepsilon_y = \left| \frac{ax^2}{ax^2} \right| \varepsilon_x = \varepsilon_x$$

$$y = a + x \quad \varepsilon_y = \left| \frac{x}{a+x} \right| \varepsilon_x$$

$$y = a - x \quad \varepsilon_y = \left| \frac{x}{a-x} \right| \varepsilon_x$$

Moltiplicazione e **divisione** sono operazioni che NON amplificano l'errore relativo.

Somma e **sottrazione** sono operazioni sicure se rispettivamente a e x hanno segno uguale o segno opposto.



Propagazione dell'errore nelle operazioni elementari

Pericolo: somma e sottrazione possono amplificare notevolmente l'errore relativo (fino a situazioni disastrose) se sono contemporaneamente soddisfatte le due condizioni:

- a e x hanno rispettivamente segno opposto o segno uguale
- $|a|$ e $|x|$ hanno molte cifre significative in comune

Esempio:

$$a = 0.12345678 \text{ e } x = 0.12345612 \rightarrow y = a - x = 0.00000066$$

se l'ottava cifra significativa di x è sbagliata, dopo la sottrazione la seconda cifra significativa di y è errata.



Propagazione dell'errore nelle operazioni elementari

N.B.: Il problema maggiore dell'analisi numerica su calcolatore è la somma di due numeri dello stesso ordine di grandezza ma di segno opposto. Il problema NON risiede nell'errore intrinseco dell'operazione ma nell'amplificazione di un preesistente errore nei dati. Se i dati sono esatti ed il calcolatore li può rappresentare senza dover eseguire arrotondamenti l'operazione non presenta problemi. In generale è sempre possibile modificare l'algoritmo di calcolo in modo da evitare la sottrazione di due numeri aventi lo stesso ordine di grandezza.

N.B.: Altro problema significativo dell'analisi numerica su calcolatore è la somma di molti numeri dello stesso segno e di ordine di grandezza decrescente.

Esempio:
$$\sum_{i=1}^{1,000,000} \frac{1}{i}$$

in questo caso è consigliabile invertire l'ordine di somma partendo dai termini più piccoli ed utilizzare la doppia precisione.



Condizionamento e Stabilità

In certe condizioni basta una piccola variazione dei dati di input o della struttura di un algoritmo per ottenere risultati significativamente (talvolta completamente) differenti.

Occorre in questi casi operare una distinzione netta e precisa tra:

- dipendenza della soluzione dai dati di input e dalla formulazione del problema. In questo caso si parla di **condizionamento del problema**.
- tipologia di algoritmo utilizzata per la risoluzione del problema numerico. In questo caso si parla di **stabilità dell'algoritmo**.



Condizionamento e Stabilità

Definizione: se il **problema reale** da risolvere numericamente è studiabile ed affrontabile a livello risolutivo si dice che il problema è **ben posto**.

Definizione: se la soluzione del **problema numerico** risulta poco dipendente dagli eventuali errori sui dati iniziali o da piccole modifiche al modello si dice che il problema è **ben condizionato**.

Definizione: un **algoritmo** è **numericamente stabile** se la sensitività alla modifica dei dati è dello stesso ordine di grandezza della sensitività del problema.

Insegnamento: si dice che un **problema** è/non è **ben posto** o è/non è **ben condizionato** ma non che è/non è stabile. Si dice che un **algoritmo** è/non è **stabile** ma non che è/non è ben posto o ben condizionato.



Condizionamento e Stabilità

Esempio: nei problemi di ingegneria chimica legati allo studio cinetico di certi sistemi reagenti spesso si desiderano studiare le condizioni di esplosività di una miscela combustibile/comburente. L'evoluzione dinamica dell'esplosione è praticamente impossibile da modellare e risolvere numericamente. Il problema numerico connesso è detto **mal posto**. È viceversa **ben posto** il sistema differenziale che studia le condizioni di ignizione che conducono all'esplosione finale del sistema.

Il moto oscillatorio del pendolo può essere modellato secondo due approcci distinti che possono rendere il problema numerico **bene** o **mal condizionato**.

La scelta di un algoritmo risolutivo piuttosto che un altro nella risoluzione ad esempio di equazioni differenziali *stiff* permette di affermare che il metodo risolutivo di Runge-Kutta IV ordine **non è stabile** mentre un algoritmo risolutivo della famiglia dei metodi di Gear è **stabile**.



Condizionamento e Stabilità

N.B.: se un problema numerico è mal posto o mal condizionato NON esiste un algoritmo in grado di risolverlo in modo affidabile.

In tal caso occorre riformulare il problema o cambiare approccio descrittivo del fenomeno reale.

La instabilità numerica è una caratteristica di un particolare schema di calcolo e non della formulazione del problema.

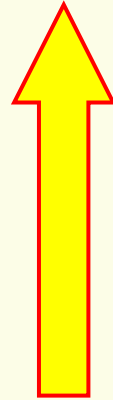
La stabilità di un algoritmo è un concetto relativo, ovvero richiede il confronto con la stabilità di un altro algoritmo.



Selezione di un algoritmo

Esistono quattro criteri gerarchicamente ordinati rispetto cui selezionare un algoritmo numerico:

1. Stabilità
2. Accuratezza
3. Tempo di calcolo
4. Occupazione di memoria



IMPORTANZA

Definizione: si definisce *flop* l'unione di un'operazione di moltiplicazione (divisione) con quella di somma (sottrazione) ed infine di assegnazione.

Esempio: $\mathbf{s} = \mathbf{s} + \mathbf{a}(\mathbf{i}, \mathbf{k}) * \mathbf{b}(\mathbf{k}) \leftarrow \text{flop}$

La potenza dei computer viene appunto misurata in flop/s (Mflop/s, Gflop/s, Tflop/s)

Lecture aggiuntive

Current 32 bit PCs have a memory limit of 4 Gigabytes RAM and in a few years PCs will be shipping with this limit.

64 bit computers will be the next step and dare I say it final one? Bill Gates (famously) thought 640 Kilobytes RAM would be enough for anyone, but he was wrong.

65,536 - 16 bit address bus e.g. BBC micro

1,048,576 - Old 16 bit (x 16 pages) IBM PCs

4,294,967,296 - 32 bit current PCs, 4 Giga-bytes

18,446,744,073,709,552,000 - 64 bit future PCs, 18 Exa-bytes

E P T G M K

Kilo Mega Giga Tera Peta Exa Zetta Yotta Xona Weka Vunda Uda Treda

Search engine google currently uses 2 Peta-bytes disc space so with 18 Exa-bytes you could fit 9000 Google's into RAM, if you could afford the RAM.

Well, if 18 Exa-bytes is still too little then try 128 bit computers with:

340,000,000,000,000,000,000,000,000,000,000

U V W X Y Z E P T G M K

340 Uda-bytes RAM memory or 170 Zetta-Google's.

It's still not that much. A person has about 7 Xona-atoms, so we can only fit in about 50 billion people to RAM.

Going up to 256 bits and RAM of 100 Tera Luma-bytes (10^{77}). There are this order of atoms in the universe so a 256 bit computer could just fit that in. Errr.... wait a minute, there wouldn't be enough atoms to build the RAM. **256 bit PCU's should be enough for anyone then.**



Extended system of units

The International system of units is a system of units to help us write down quantities without having to use scientific notation or lots of digits or zeroes. For example, a million Hertz is a megahertz; a billionth of a meter is a nanometer, and a billion meters is a gigameter, which I have almost never heard of; it is 1/4 of the way to the Moon. The international system first invented a system of prefixes that went to 10^{12} and to 10^{-18} . This proved to be inadequate; for example, we are now talking about thousands of terabytes. So this was extended to 10^{18} , the small units still went only to 10^{-18} . Even this has proven to be inadequate. So the scientific system went two steps farther in each direction. 10^{21} is zetta-, and 10^{24} is yotta-, and for the first time this week, I saw yotta- actually being used. The sun puts out 380 yottawatts of power, says Nearest Star, by Leon Golub and Jay M. Pasachoff, page 12. I can see how these endings were derived. zetta is z + -etta, which is an alteration of septi-, meaning 7, as 21 is 7 groups of three. yotta is y + -otta, an alteration of octo-, meaning 8. The pattern here is that we go backwards from the beginning of the alphabet, starting with z and y, and we follow it up with an alteration of the Greek or Latin for the next number. According to this pattern, the next ending should be xona-, since x comes before y in the alphabet, and 9 is noni- in Latin. Similarly, 10^{30} should be weka-, since w precedes x and 10 is deka in Greek. Here is my proposal for extending the system all the way to 10^{63} , a vigintillion, the highest *continuous* -illion number recognized in dictionaries. Why way out this far? We talk of bigger and smaller things all the time. For example, a jiffy is 1/10 of a rimotosecond, 10^{-43} second, which some say is the smallest unit of time possible.

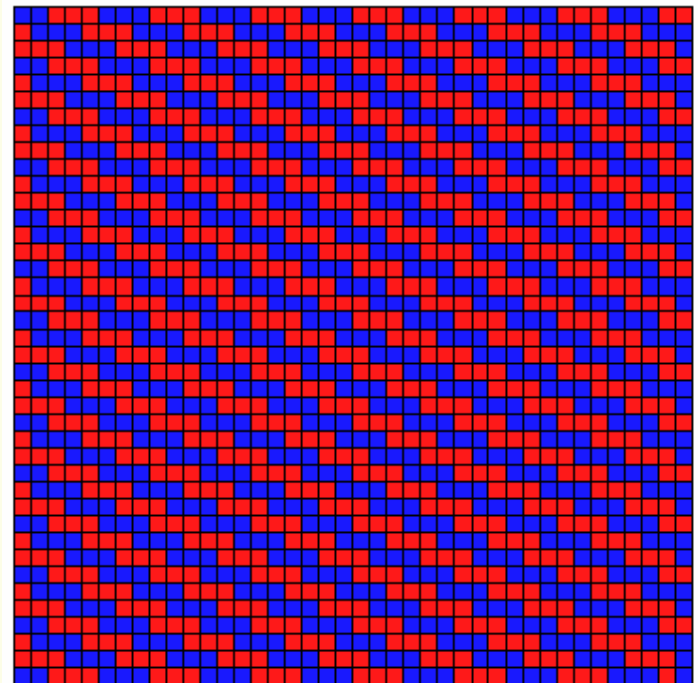
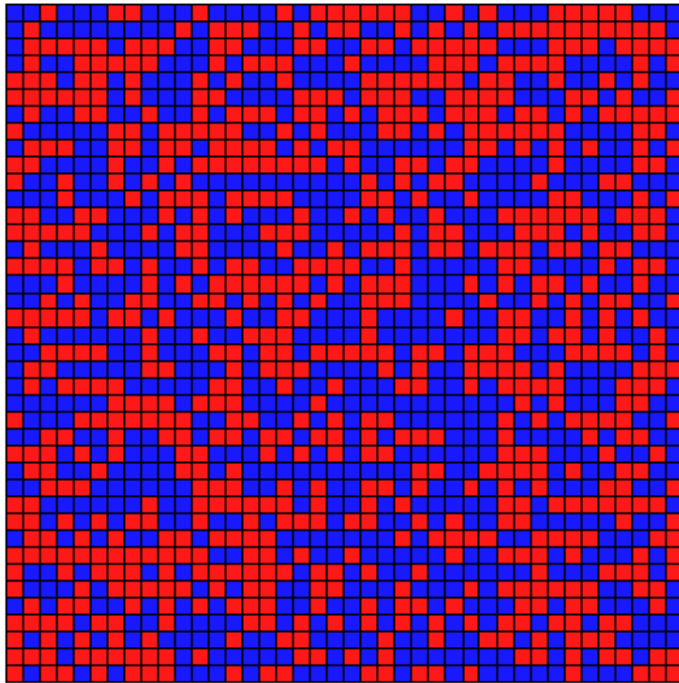


Extended system of units

Factor	Name	Symbol	Factor	Name	Symbol
10^1	deka	da	10^{-1}	deci	d
10^2	hecto	h	10^{-2}	centi	c
10^3	kilo	k	10^{-3}	milli	m
10^6	mega	M	10^{-6}	micro	μ
10^9	giga	G	10^{-9}	nano	n
10^{12}	tera	T	10^{-12}	pico	p
10^{15}	peta	P	10^{-15}	femto	f
10^{18}	exa	E	10^{-18}	atto	a
10^{21}	zetta	Z	10^{-21}	zepto	z
10^{24}	yotta	Y	10^{-24}	yocto	y
10^{27}	xona	X	10^{-27}	xonto	x
10^{30}	weka	W	10^{-30}	wekto	w
10^{33}	vunda	V	10^{-33}	vunkto	v
10^{36}	uda	U	10^{-36}	unto	u
10^{39}	treda	TD	10^{-39}	trekto	td
10^{42}	sorta	S	10^{-42}	sotro	s
10^{45}	rinta	R	10^{-45}	rimto	r
10^{48}	quexa	Q	10^{-48}	quekto	q
10^{51}	pepta	PP	10^{-51}	pekro	pk
10^{54}	ocha	O	10^{-54}	otro	o
10^{57}	nenas	N	10^{-57}	nekto	nk
10^{60}	minga	MI	10^{-60}	mikto	mi
10^{63}	luma	L	10^{-63}	lunto	l



Rappresentazione binaria di π e di $22/7$

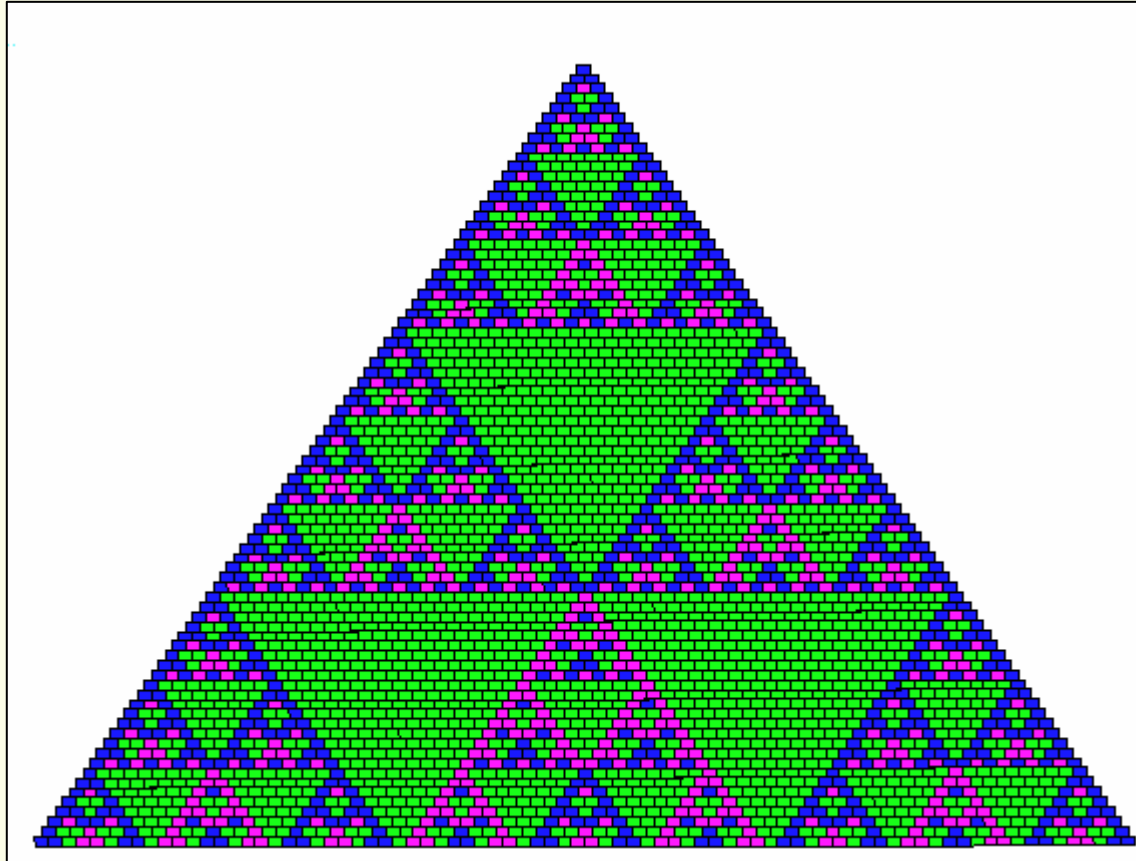


Dall'alto in basso e da sinistra a destra (come per leggere un libro) sono rappresentate le prime 1600 cifre di π e di una sua approssimazione razionale $22/7$.

In rosso le cifre pari ed in blu quelle dispari. Si noti l'assoluta mancanza di ripetizione o di strutturazione del numero irrazionale π .

Nel diciassettesimo secolo Gottfried Wilhelm Leibniz pose il problema, in una lettera ad uno dei fratelli Bernoulli, se esistesse una qualche struttura ripetitiva (pattern) nella rappresentazione binaria delle cifre di π .

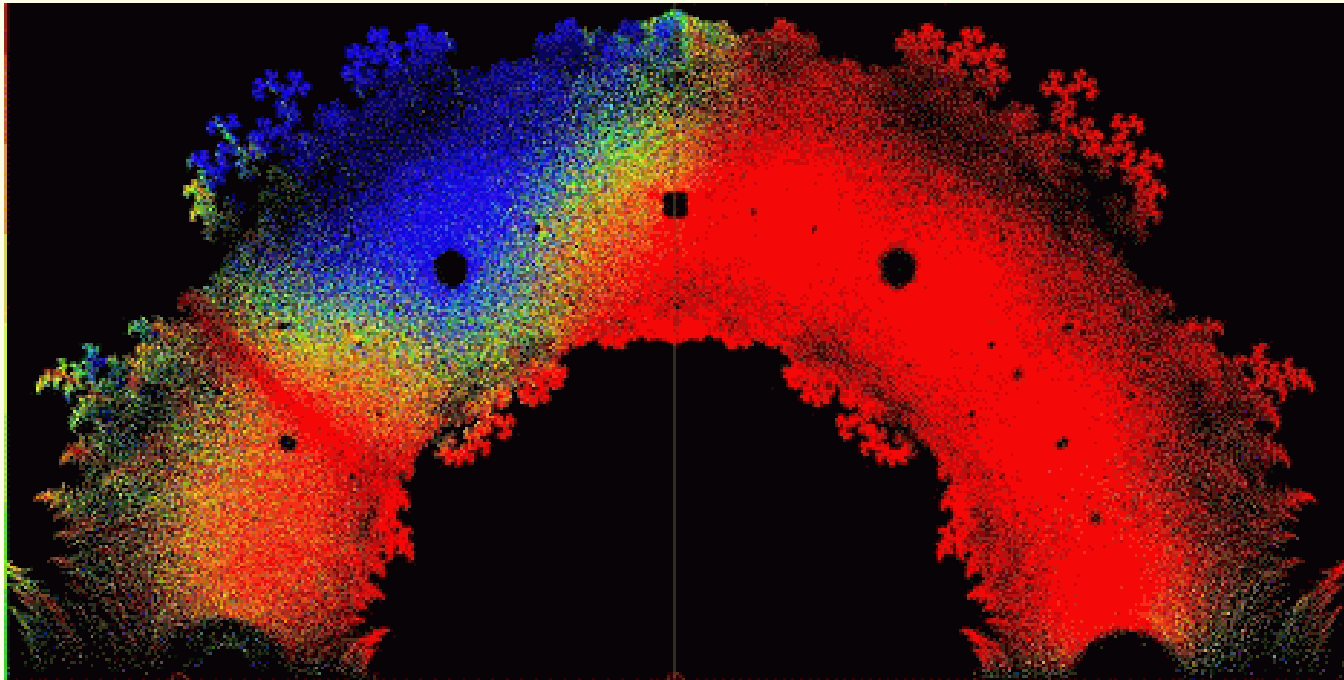
La struttura dei numeri



Questa figura mostra i coefficienti del triangolo di Pascal (triangolo di Tartaglia) in modulo 3. Equivalentemente si può pensare ad ogni i -esima riga del triangolo come composta dai coefficienti del polinomio $(1-x)^i$ in modulo 3 (resto della divisione intera per 3).

Esempio: $16 \text{ Mod } 3 = 1$, $18 \text{ Mod } 4 = 2$, $20 \text{ Mod } 5 = 0$.

Zeri complessi di un polinomio



Questa figura mostra gli zeri di un polinomio della forma: $P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$

dove $n \leq 18$, $a_i = \{-1, \dots, +1\}$ secondo come si dispongono sul piano complesso.

Sorgono spontanee alcune domande: le soluzioni costituiscono un insieme frattale? Continuano a persistere dei "buchi neri" quando l'ordine del polinomio tende ad infinito? Come variano i "buchi neri" con l'ordine del polinomio?

Sui numeri floating point in formato IEEE

No.	Parameter	Format					
		Single	Single Extended	Double	Double Extended	Quadruple ⁺	Extended [#]
(1)	p (precision, apparent mantissa width in bits)	24	≥ 32	53	≥ 64	113	64
(2)	Decimal digits of precision $p / \log_2(10)$	7.22	≥ 9.63	15.95	≥ 19.26	34.01	19.26
(3)	Mantissa's MS-Bit	hidden bit	unspecified	hidden bit	unspecified	hidden bit	explicit bit
(4)	Actual mantissa width in bits	23	≥ 31	52	≥ 63	112	64
(5)	E_{\max}	+127	$\geq +1023$	+1023	$\geq +16383$	+16383	+16383
(6)	E_{\min}	-126	≤ -1022	-1022	≤ -16382	-16382	-16382
(7)	Exponent <i>bias</i>	+127	unspecified	+1023	unspecified	+16383	+16383
(8)	Exponent width in bits	8	≥ 11	11	≥ 15	15	15
(9)	Sign width in bits	1	1	1	1	1	1
(10)	Format width in bits (9) + (8) + (4)	32	≥ 43	64	≥ 79	128	80
(11)	Range Magnitude Maximum $2^{E_{\max} + 1}$	3.4028E+38	$\geq 1.7976E+308$	1.7976E+308	$\geq 1.1897E+4932$	1.1897E+4932	1.1897E+4932
(12)	Range Magnitude Minimum $2^{E_{\min}}$	1.1754E-38	$\leq 2.2250E-308$	2.2250E-308	$\leq 3.3621E-4932$	3.3621E-4932	3.3621E-4932
(13)	Range Magnitude Minimum (Denormalized) $2^{E_{\min} - (4)}$	1.4012E-45	$\leq 1.0361E-317$	4.9406E-324	$\leq 3.6451E-4951$	6.4751E-4966	1.8225E-4951
(14)	FORTRAN Language Type	REAL*4		REAL*8		REAL*16	REAL*10
(15)	C Language Type	float		double		long double	long double

© Copyright 1985 by The Institute of Electrical and Electronics Engineers, Inc

<http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html#history>



Sui numeri floating point in formato IEEE

IEEE Floating-Point Values

Name	Quantity	Exponent	Significand
Signed zero	± 0	$E = E_{\min} - 1$	$sig = 0$
Denormalized number	$\pm 0 . sig \times 2^{E_{\min}}$	$E = E_{\min} - 1$	$sig \neq 0$
Normalized number	$\pm 1 . sig \times 2^E$	$E_{\min} \leq E \leq E_{\max}$	sig
Signed infinity	$\pm \infty$	$E = E_{\max} + 1$	$sig = 0$
Not a Number	NaN	$E = E_{\max} + 1$	$sig \neq 0$

These special values are interpreted as follows:

- Signed zero

Fortran PowerStation treats zero as signed by default. The sign of zero is the same as the sign of a non-zero number. If you use the intrinsic function **SIGN** with zero as the second argument, the sign of the zero will be transferred. Comparisons, however, consider +0 to be equal to -0. A signed zero is useful in certain numerical analysis algorithms, but in most applications the sign of zero is invisible.

- Denormalized numbers

Denormalized numbers (denormals) fill the gap between the smallest positive number and the smallest negative number. Otherwise only (\pm) 0 occurs in that interval. Denormalized numbers permit gradual underflow for intermediate results calculated internally in extended-double format. Fortran PowerStation sets a status flag when a number loses precision due to denormalization.



Sui numeri floating point in formato IEEE

- Signed infinity

Infinities are the result of arithmetic in the limiting case of operands with arbitrarily large magnitude. They provide a way to continue when an overflow occurs. The sign of an infinity is simply the sign you obtain for a finite number in the same operation as the finite number approaches an infinite value. By retrieving the status flags described in [Setting and Retrieving Floating-Point Status and Control Words](#) in this section, you can differentiate between an infinity that results from an overflow and one that results from division by zero. Fortran PowerStation treats infinity as signed by default. The output value of infinity is $\pm 1\#INF$.

- Not a Number

Not a Number (NaN) results from an operation involving one or more invalid operands. For instance $0/0$ and $0(-1)$ result in NaN. In general, an operation involving a NaN produces another NaN. Because the fraction of a NaN is unspecified, there are many possible NaNs. Fortran PowerStation treats all NaNs identically. The output value of NaN is $1\#IND$.

N.B.: dalle note precedenti è possibile evincere come mai lavorando in singola precisione si ha $E_{min} = -126$ ed $E_{max} = +127$. Dato che i bit disponibili per l'esponente sono 8 si dovrebbero avere 256 alternative differenti (ad esempio tra -128 e $+127$, oppure tra -127 e $+128$). In realtà all'appello sembrano mancare due stati dato che $127 + 126 + 1$ (per lo zero) = 254 anziché 256. I due stati mancanti (ovvero un bit (0, 1)) sono dedicati al bit che serve per identificare il valore NaN o Inf. Si noti come, giocando con la mantissa (significand) e con $E_{max}+1$ o $E_{min}-1$, sia possibile infine distinguere tra 4 stati differenti: zero con segno, underflow con segno (denormalized), NaN e Inf.



Sui numeri floating point in formato IEEE

```
SUBROUTINE CPUNumbersSingle
  implicit real*4 (a-h,o-z)          !          Ovvero SINGOLA PRECISIONE
```

```
  eps = 1.
  somma = 1. + eps
  do while(somma > 1.)
    eps = 0.5 * eps
    somma = 1. + eps
  enddo
  eps = 2. * eps
  xTiny = 2.**(-127)
  xHuge = 2.**((1.-2.**(-23))*2.**127)
```

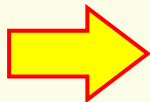
$$tiny = 2^{L-1}$$

$$huge = (1 - 2^{-P}) 2^U$$

```
  write(*,*)' Macheps in single precision      = ',eps
  write(*,*)' Macheps in single precision bis = ',2.**(-23)
  write(*,*)' Tiny in single precision         = ',xTiny
  write(*,*)' Tiny in single precision bis     = ',Tiny(1.2345)
  write(*,*)' Huge in single precision         = ',xHuge
  write(*,*)' Huge in single precision bis     = ',Huge(1.2345)
  write(*,*)' *****'
  return
```

END

```
  Macheps in single precision      =      1.192093E-07
  Macheps in single precision bis =      1.192093E-07
  Tiny in single precision         =      5.877472E-39
  Tiny in single precision bis     =      1.175494E-38
  Huge in single precision         =      3.402823E+38
  Huge in single precision bis     =      3.402823E+38
```



Sui numeri floating point in formato IEEE

SUBROUTINE CPUNumbersDouble

implicit real*8 (a-h,o-z)

!

Ovvero DOPPIA PRECISIONE

eps = 1.d0

somma = 1.d0 + eps

do while(somma > 1.d0)

eps = 0.5d0 * eps

somma = 1.d0 + eps

enddo

eps = 2.d0 * eps

xTiny = 2.d0**(-1022-1)

xHuge = 2.d0*((1.d0-2.d0**(-52))*2.d0**1023)

write(*,*)' Macheps in double precision = ',eps

write(*,*)' Macheps in double precision bis = ',2.d0**(-52)

write(*,*)' Tiny in double precision = ',xTiny

write(*,*)' Tiny in double precision bis = ',Tiny(1.2345d0)

write(*,*)' Huge in double precision = ',xHuge

write(*,*)' Huge in double precision bis = ',Huge(1.2345d0)

write(*,*)' *****'

return

END

Macheps in double precision = 2.220446049250313E-016

Macheps in double precision bis = 2.220446049250313E-016

Tiny in double precision = 1.112536929253601E-308

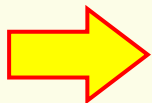
Tiny in double precision bis = 2.225073858507201E-308

Huge in double precision = 1.797693134862316E+308

Huge in double precision bis = 1.797693134862316E+308

$$tiny = 2^{L-1}$$

$$huge = (1 - 2^{-P})2^U$$



Sui numeri floating point in formato IEEE

A pagina L-24 viene detto che $3.576279\text{E}-07$ è il primo numero che si distingue da 3. Sommando cioè tale numero a 3 si ottiene un nuovo numero che è diverso da 3.

Il seguente programma in Fortran mette in evidenza come in realtà si possa ridurre ulteriormente tale valore ottenendo secondo il computer un numero che è ancora differente da 3.

```
PROGRAM PrecisioneMacchina
! Calcoli di processo dell'ingegneria Chimica
! Politecnico di Milano
! Davide Manca
! Vedi pag. L2-24
! 19-Oct-2005 18:05:34
!-----
! Macheps secondo la definizione IEEE per singola precisione a 32 bit
xMEPSingle = 2.**(-23)
write(*,*)'xMEPSingle = ',xMEPSingle

fattoreMoltiplicativo = 1.0
epsi = 3. * xMEPSingle * fattoreMoltiplicativo
write(*,*)'epsi = ',epsi
xSingle = 3. + epsi
diff = xSingle - 3.

if(xSingle == 3.)then
  write(*,1)'I due numeri sono uguali !!! ',xSingle,diff
else
  write(*,1)'I due numeri sono diversi !!! ',xSingle,diff
endif

1 format(/1x,A,2x,f20.16,2x,e24.16)

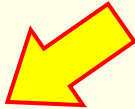
END
```



Sui numeri floating point in formato IEEE

Eseguendo il programma "PrecisioneMacchina", riportato alla pagina precedente, si ottengono i seguenti risultati:

```
fattoreMoltiplicativo = 1.  
xMEPSingle = 1.192093E-07  
epsi = 3.576279E-07  
  
I due numeri sono diversi !!! 3.0000000000000000 .3576279000000000E-06
```




È possibile osservare che il numero che dovrebbe per primo perturbare 3 è proprio $3.576279E-07$.

In realtà, giocando con la variabile: `fattoreMoltiplicativo` e facendola decrescere da 1 fino a 0.4 si ottengono risultati che indicano che `xSingle` e 3. sono ancora due numeri differenti.

Soltanto quando `fattoreMoltiplicativo` è pari a 0.3 il programma afferma che i due numeri sono uguali!

```
fattoreMoltiplicativo = 0.3  
xMEPSingle = 1.192093E-07  
epsi = 1.072884E-07  
  
I due numeri sono uguali !!! 3.0000000000000000 .1072884000000000E-06
```



Si noti peraltro che la scrittura a video del valore `xSingle` appare identicamente uguale a 3 in entrambi i casi.

Ponendo viceversa `fattoreMoltiplicativo = 2`. Si ottiene:

```
xSingle = 3.0000010000000000
```


Bibliografia

- ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, (1985)
- Atkinson K. E., "Elementary Numerical Analysis", John Wiley & Sons, (1993)
- Buzzi Ferraris G., "Metodi numerici e Software in C++", Addison Wesley, (1998)
- Comincioli V., "Analisi numerica. Metodi Modelli Applicazioni", McGraw-Hill, (1990)
- Gamow G., "Trent'anni che sconvolsero la fisica", Zanichelli, (1966)
- Saracco R., "La memoria del futuro", Apogeo, (2002)

- <http://www.aip.org/history/einstein/>
- <http://www.aip.org/history/heisenberg/>
- <http://www.cecm.sfu.ca/~loki/Papers/Numbers/>
- <http://www.nobel.se/physics/educational/transistor/history/>
- http://www.plexos.com/256_bit_CPUs_should_be_enough.htm
- <http://digilander.libero.it/lucianobattaia/>
- <http://vt100.net/docs/misc/core/>
- http://docs.sun.com/source/806-3568/ncg_goldberg.html
- <http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF>
- <http://stevehollasch.com/cgindex/coding/ieeefloat.html>

