

Giovanni Di Maria – Gruppo Eratostene Caltanissetta

(con la partecipazione dell'Ing. Rosario Turco)

I metodi per calcolare il Logaritmo

DIC 2008 – (prima revisione SET 2008)

Nomenclatura

LN(x)	è il logaritmo naturale
LOG(x)	è il logaritmo a base 10
LG(x)	è il logaritmo a base 2
E	è la costante di Nepero che equivale a 2.718281828459
LN(x)	= LOG(x) / LOG(E) oppure LG(x) / LG(E)
LOG(x)	= LN(x) / LN(10) oppure LG(x) / LG(10)
LG(x)	= LN(x) / LN(2) oppure LOG(x) / LOG(2)

I metodi per calcolare il Logaritmo

Con la tecnologia attuale, il calcolo del logaritmo di un numero è cosa semplice. Basta disporre di una calcolatrice scientifica o di un computer, ed il gioco è fatto. Qualche decennio fa si utilizzavano le tavole logaritmiche, che consistevano in una successione molto grande di dati precalcolati, con cui era anche semplice calcolare il logaritmo di un numero.

Eppure ci sono dei casi in cui si deve effettuare il calcolo del logaritmo senza effettivamente disporre della sua funzione diretta, ossia occorre calcolarlo “a mano” con l’ausilio delle quattro operazioni fondamentali, più qualcuna aggiuntiva, come l’elevamento a potenza o la radice quadrata.

Questo avviene, ad esempio, se si deve implementare la funzione in un sistema sprovvisto di calcolo di logaritmo, come ad esempio un linguaggio di programmazione semplice.

Nell’articolo saranno esaminati molti metodi per il calcolo di questa importante funzione, seguiti anche da numerosi esempi pratici per comprenderne a fondo il funzionamento, in modo da effettuarne successivamente la relativa implementazione nel sistema.

Sarà data molta importanza alla possibilità di realizzazione di algoritmi per l’utilizzo meccanizzato delle procedure su elaboratori elettronici.

Metodo diretto (a cura di Giovanni Di Maria)

Il metodo più semplice ed immediato è quello di utilizzare le funzioni appropriate nelle calcolatrici e nei linguaggi di programmazione. La maggior parte dei sistemi mettono a disposizione le funzioni per il rapido calcolo, soprattutto in base 10 e in base E. Con le equazioni riportate nella nomenclatura è tuttavia possibile calcolare il logaritmo in qualsiasi base, con una semplice divisione.

Formula di Borchardt (a cura di Giovanni Di Maria)

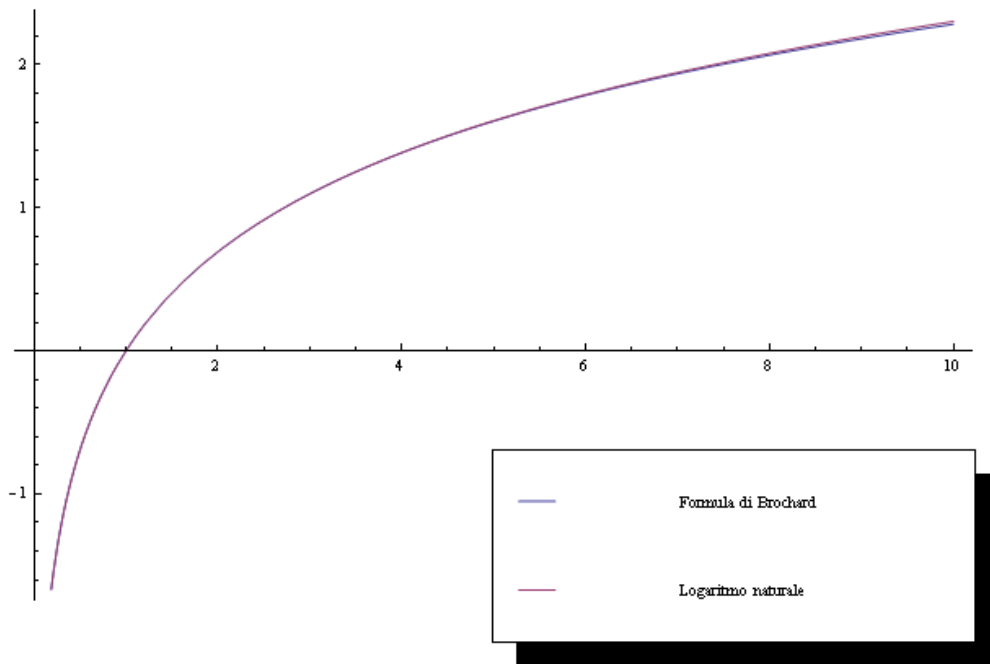
Si tratta di una elegante e comoda formula, per lo più immediata, utile quando i valori su cui trovare il **logaritmo naturale** sono bassi. La funzione è infatti abbastanza precisa per valori di x fino a 20, ma essa comincia a divergere e a perdere significato utile quando le quantità cominciano a salire.

La formula è la seguente:

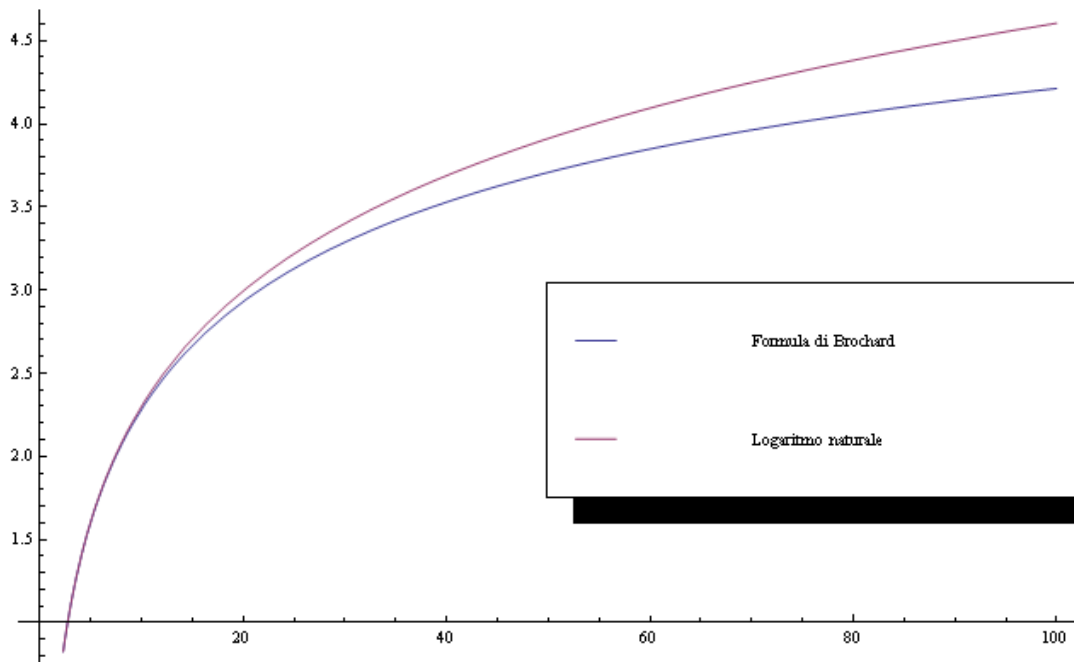
$$\ln(x) \approx \frac{6(x-1)}{x + 4x^{0.5} + 1}$$

Si può implementare su qualsiasi linguaggio di programmazione poiché, come si osserva, essa prevede solo la sottrazione, l'addizione, la divisione, la moltiplicazione e l'elevamento a potenza. Occorre però, come detto, utilizzare piccoli valori noti.

I metodi per calcolare il Logaritmo – Giovanni Di Maria



Andamento della funzione per piccoli valori di x



Andamento della funzione per valori più grandi di x

Espansione delle Serie (a cura di Giovanni Di Maria)

Molte funzioni reali, tra cui quella del logaritmo naturale, sono sviluppabili attraverso le serie di Mac Laurin. In questa trattazione interessa solamente quella per calcolare il logaritmo neperiano di un qualunque numero.

L'unico problema dell'espansione delle serie, specialmente di questo tipo, è dato dalla lentezza intrinseca del metodo, in quanto essa converge molto lentamente e, per alcuni casi, vengono coinvolti risultati parziali molto grandi. Inoltre il dominio di convergenza risulta alquanto limitato.

Proponiamo tre serie diverse, secondo il dominio della variabile cercata.

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots - \frac{(-x)^n}{n} \quad (\text{per } -1 < x < 1)$$

$$\ln(x) = \left(\frac{x-1}{x}\right) + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 \quad (\text{per } x \geq \frac{1}{2})$$

$$\ln\left(\frac{x+1}{x-1}\right) = 2\left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \frac{1}{7x^7} + \dots\right) \quad (\text{per } |x| \geq 1)$$

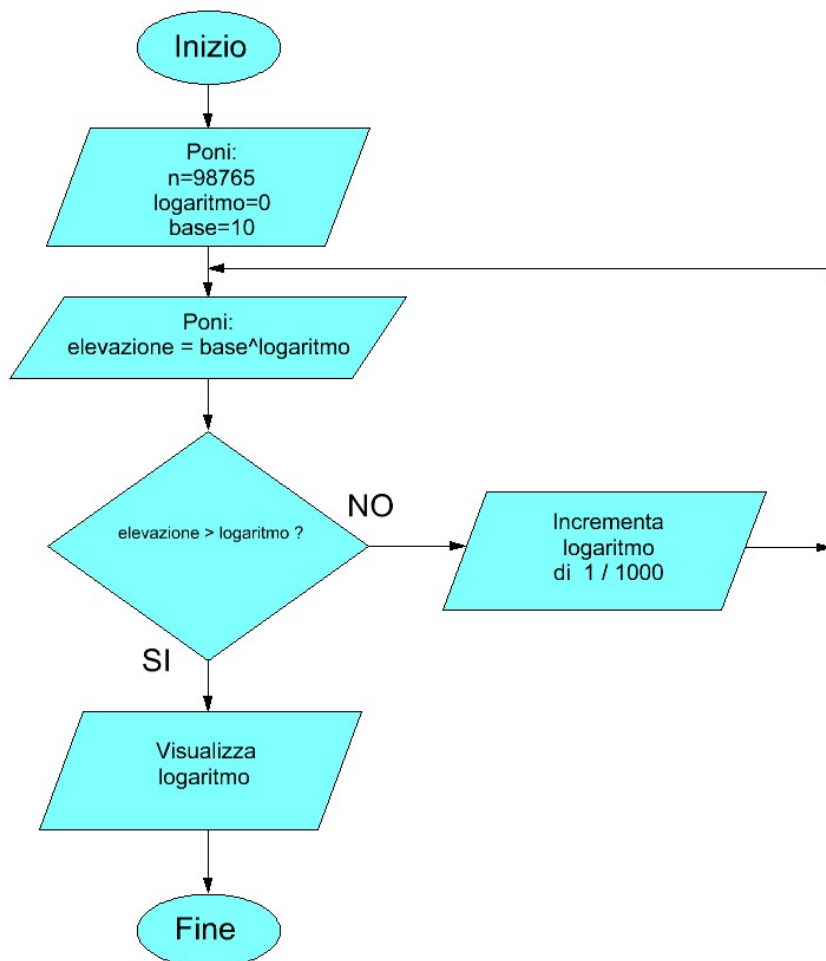
ovvero

$$\ln\left(\frac{x+1}{x-1}\right) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots\right) \quad (\text{per } |x| \geq 1)$$

Metodo di avvicinamento (di Giovanni Di Maria)

Questo metodo si attua tramite un algoritmo. La sua filosofia sta nel rintracciare il valore del logaritmo di un numero mediante avvicinamento. Normalmente, il valore simulato parte da zero e incrementa di uno specifico passo, fino a raggiungere il risultato. Più piccolo è il passo e più preciso risulterà il risultato finale, anche se occorrerà maggiore tempo per la sua conclusione.

Il funzionamento del metodo, tramite un esempio di diagrammi a blocchi, è il seguente:



Di seguito è mostrato il listato di programma, scritto in linguaggio C, che offre una velocità maggiore di ogni altro linguaggio:

```
#include <stdio.h>
#include <math.h>
void main() {
    float n,logaritmo,base,elevazione;
    n=98765;
    logaritmo=0;
    base=10;
    while (1) {
        elevazione=pow(base,logaritmo);
        if(elevazione>n)
            break;
        logaritmo+=0.001;
    }
    printf("Il logaritmo e' %f\n",logaritmo);
}
```

Se occorrono valori diversi per l'operando o per la base, basta semplicemente modificare rispettivamente i valori della variabile n e base.

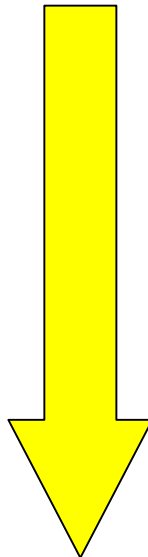
Metodo dicotomico (a cura di Giovanni Di Maria)

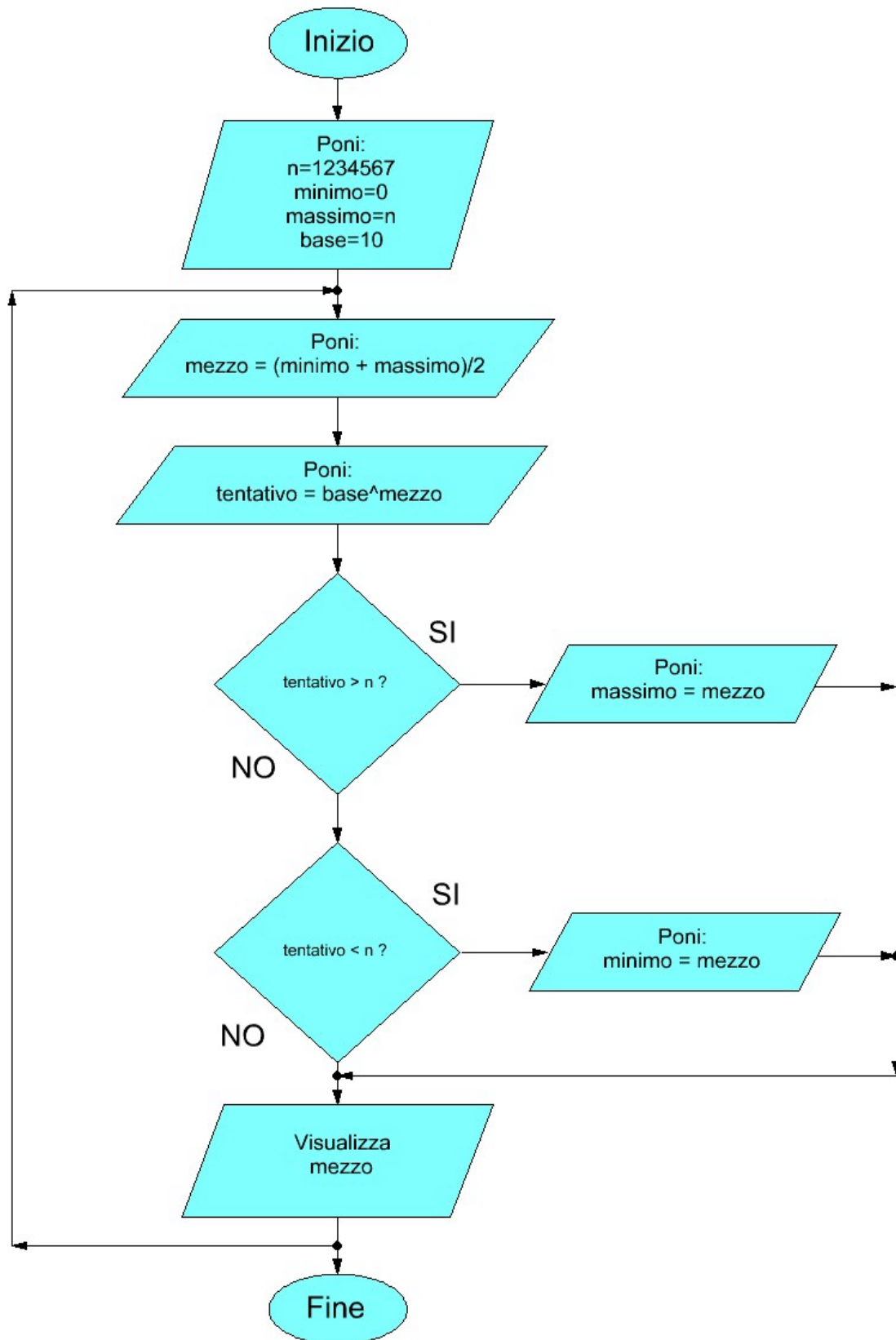
Anche questo metodo si risolve tramite un algoritmo. Il tempo di ricerca è sicuramente inferiore a quello precedente, in quanto il valore del logaritmo è cercato con una strategia più intelligente.

In pratica, il rintracciamento del valore non avviene sequenzialmente. Vengono invece posti due limiti (uno inferiore e uno superiore, inizialmente impostati a valori estremi) ed il risultato viene cercato nel punto centrale di essi, ossia nella media aritmetica dei due limiti.

I due limiti si avvicinano sino a trovare il risultato, a seconda che il calcolo di verifica sia maggiore o minore di quello cercato.

Il diagramma a blocchi di esempio esplicherà maggiormente l'idea:





Di seguito è mostrato il listato di programma, scritto in linguaggio C, che offre una velocità maggiore di ogni altro linguaggio:

```
#include <stdio.h>
#include <math.h>
void main() {
    float n,minimo,massimo;
    float base,mezzo,tentativo;
    n=1234567;
    minimo=0;
    massimo=n;
    base=10;
    while (1) {
        mezzo=(minimo+massimo)/2;
        tentativo=pow(base,mezzo);
        if(tentativo>n)
            massimo=mezzo;
        if(tentativo<n)
            minimo=mezzo;
        printf("Il logaritmo e' %f\n",mezzo);
    }
}
```

Se servono valori diversi per l'operando o per la base, basta semplicemente modificare rispettivamente i valori della variabile n e base.

Impostato in questo modo, il programma non termina mai, ma visualizza, passo dopo passo, il valore del logaritmo che si va avvicinando a quello reale, ad ogni iterazione del programma.

Metodo formula inversa (a cura di Giovanni Di Maria)

Per la risoluzione a formula inversa, è necessario risolvere il sistema seguente:

$$\begin{cases} \text{Log}_b(x) = n \\ b^n = x \end{cases}$$

Esempio:

$$\begin{cases} \text{LOG}(100) = 2 \\ 10^2 = 100 \end{cases}$$

Metodo delle elevazioni (a cura di Giovanni Di Maria)

Si tratta di un metodo **semplice** e **geniale** che può essere eseguito anche tramite una calcolatrice non molto sofisticata e che può essere implementato in un algoritmo per qualunque linguaggio di programmazione sprovvisto della funzione per il calcolo del logaritmo.

Per attuare il metodo è sufficiente eseguire alcune elevazioni e alcuni conteggi.

Con un esempio pratico viene spiegato il metodo. Si debba calcolare il logaritmo di 150 in base 10.

Si divida 150 per 10 fino a quando il risultato è maggiore di 10. Si totalizzino quindi il numero delle divisioni. Il conteggio costituirà la parte intera del risultato.

$$\begin{array}{ll} 150 : 10 = 15 & 1^\circ \\ 15 : 10 = 1,5 & 2^\circ \quad \leftarrow \end{array}$$

Non potendo effettuare più divisioni, il risultato intero sarà 2, ossia il numero di divisioni.

Adesso si possono calcolare le cifre decimali, con una precisione pressoché infinita.

Si proceda in questo modo: si elevi a 10 il risultato finale (1,5) e si proceda a dividere per 10 fino a quando la divisione lo consenta. Allo stesso modo si totalizzino il numero di divisioni. Tale risultato costituirà la prima cifra decimale:

$$\begin{array}{ll} 1,5^{\wedge} 10 = 57,66503906 & \\ 57,66503906 : 10 = 5,766503906 & 1^\circ \quad \leftarrow \end{array}$$

Non potendo effettuare più divisioni, la prima cifra decimale sarà 1, ossia il numero di divisioni.

Adesso si può calcolare la seconda cifra decimale. Si proceda in questo modo: si elevi a 10 il risultato finale (5,766503906) e si proceda a dividere per 10 fino a quando la divisione lo consenta. Allo stesso modo si totalizzino il numero di divisioni. Tale risultato costituirà la seconda cifra decimale:

$$\begin{aligned} 5,766503906 \wedge 10 &= 40656117,75 \\ 40656117,75 : 10 &= 4065611,775 & 1^\circ \\ 4065611,775 : 10 &= 406561,1775 & 2^\circ \\ 406561,1775 : 10 &= 40656,11775 & 3^\circ \\ 40656,11775 : 10 &= 4065,611775 & 4^\circ \\ 4065,611775 : 10 &= 406,5611775 & 5^\circ \\ 406,5611775 : 10 &= 40,65611775 & 6^\circ \\ 40,65611775 : 10 &= 4,065611775 & 7^\circ \quad \leftarrow \end{aligned}$$

Procedendo in questo modo si possono calcolare tante cifre decimali, con una normale calcolatrice.

Raccogliendo il numero di divisioni effettuate il risultato è il seguente:

$$\text{LOG}(150)=2,17\dots\dots$$

Non sono a conoscenza se tale metodo esista. In rete non ho trovato nulla in merito. Comunque l'ideazione della procedura è frutto di mie prove e miei esperimenti numerici.

Segue il listato della procedura in linguaggio C con l'implementazione delle librerie GMP, per le quali è necessaria l'inclusione in fase di compilazione. Con tali librerie è possibile effettuare il calcolo anche per numeri con numero arbitrario di cifre, superando quella che è la precisione hardware del processore.

```
double logaritmo10(mpz_t n,unsigned char precisione) {
    /*=====DICHIARAZIONI=====*/
    double logaritmo;
    char num_divisioni;
    mpf_t dividendo;
    int compara,k;
    /*=====INIZIALIZZAZIONI=====*/
    logaritmo=0;
    mpf_init(dividendo);
    mpf_set_z(dividendo,n);
    /*=====CALCOLO CIFRA PER CIFRA=====*/
    for(k=1;k<=precisione;k++) {
        num_divisioni=0;
        while(1) {
            compara=mpf_cmp_ui(dividendo,10);
            if(compara==0 || compara<0)
                break;
            mpf_div_ui(dividendo,dividendo,10);
            num_divisioni++;
        }
        logaritmo=logaritmo+((double)num_divisioni/(double)(pow(10,k-1)));
        mpf_pow_ui(dividendo,dividendo,10);
    }
    return(logaritmo);
}
```

Metodo dello shift (a cura dell'ing. Rosario Turco)

Leggendo e ragionando sugli algoritmi per i logaritmi, in realtà ce n'è uno facile-facile e rapido basato sugli shift dei bit di più basso livello e con cui calcolare qualsiasi logaritmo in base qualsiasi.

Sappiamo che passare da una base ad un'altra si può fare in questo modo:

$$\log_{b_1}(x) = \log_{b_2}(x) / \log_{b_2}(b_1)$$

Ad esempio

$$\text{Log}_e(10) = \log_2(10) / \log_2(e)$$

Ora un algoritmo semplice per il logaritmo in base 2 che sfrutta gli shift di un intero (visto come binario) è:

```
/* Funzione per calcolare il logaritmo in base 2 di un intero */
int log2(int N){
    int k=N, i=0;
    while(k){
        k>>=1; // shift di bit
        i++;
    }
    return i-1;
}
```

Alla fine si deve applicare il cambio di base, ed il calcolo è fatto, senza usare interpolazioni. Ovviamente la tecnica è usabile con linguaggi che consentono lo shift di bit.

Metodo in Pari/GP (a cura dell'ing. Rosario Turco)

Algoritmo che implementa un logaritmo in base qualsiasi

L'ideazione di un algoritmo che implementi il logaritmo si basa su due elementi di base:

- uno matematico, ovvero un logaritmo in qualsiasi base b si può ricavare a partire da uno in base x nota con la formula:

$$(1) \log_b(n) = \log_x(n) / \log_x(b)$$

- uno informatico, dipendente dagli strumenti disponibili nel linguaggio.

Se è disponibile lo shift di bit, essa è una operazione di basso livello del linguaggio e rapida.

In PARI/GP lo è (funzionalità shift), in C/C++ anche con i simboli << (shift a sinistra) e >> shift a destra.

In tal caso si dispone come operazione proprio il log in base 2 (in base $x=2$).

Ad esempio se abbiamo $n=1$ in binario è 001; in un binario ogni cifra ha peso potenza di 2. Ad esempio

$$001 = 1*2^0 + 0*2^1 + 0*2^2 = 1$$

se facciamo lo shift a sinistra $001 \ll$ per 2 volte significa che il binario diventa 100 in realtà abbiamo moltiplicato per potenze di 2

$$100 = 0*2^0 + 0*2^1 + 1*2^2 = 4$$

Se il 4 in binario lo shiftiamo a destra $100 \gg$ otteniamo che lo dividiamo per potenze di 2, ma in pratica stiamo cercando l'esponente a cui elevare la base 2 del potenza che mi ritorni 4 ovvero stiamo cercando il log in base 2 e il numero di shift che devo fare è il valore del log in base 2 ($\log_2(4)=2$).

Però dovendo cercare un criterio per arrestare l'algoritmo, si shifta a destra il numero (con PARI/GP è $\text{shift}(n,-1)$) finchè non diventa 0. In tal caso il numero di shift è di una unità superiore al valore del log in base 2.

n non potenza di 2 - precisione nel calcolo

L'algoritmo di sopra funziona se il numero n di cui calcolare l'algoritmo è effettivamente una potenza di 2.

Ad esempio 3,5,6 e 7 non lo sono. Il log in base 2 di 3 ad esempio è compreso tra gli esponenti di 2^1 e 2^2 quindi tra 1 e 2.

Nel caso di n non potenza di 2, sapendo il valore dell'estremo superiore dall'algoritmo degli shift (shift che mi da l'esponente della potenza 2 però maggiore di un'unità), allora si decrementa l'estremo con un numero che rappresenta la precisione che vogliamo ottenere, ad esempio 0.00001 .

Maggiore è tale numero, maggiore è l'attesa. Un numero a 5 cifre è comunque accettabile come precisione.

L'estremo è decrementato finchè $2^{\text{estremo}} > n$. Se siamo sotto o uguale abbiamo trovato un valore che va incrementato di 1.

Basta fare qualche prova.

Arrotondamento

A causa del decremento di 0.00001, il $\log_2(32) = 4.99999$ e non è 5. D'altra parte esistono dei $\log_2(n)$ che sono con virgola mobile giustamente ad esempio $\log_2(10) = 3.32192$ sarebbe errato approssimarlo a 4.0.

Allora in prima istanza si arrotonda al valore superiore poi si controlla se il valore superiore è maggiore del calcolo dello shift + l'arrotondamento, se lo è non si deve approssimare al valore superiore.

Logaritmo in qualsiasi base

Disponendo del \log_2 si può calcolare con la (1) qualsiasi logaritmo.

Logaritmo in base neperiana

L'ultimo inghippo con la (a) è calcolare il $\log_2(e)$ però è semplice usare la costante $e=2.71828182$.

Vantaggi

- Semplicità algoritmica;
- Velocità;
- Nessuna interpolazione;
- Precisione fino alla 5 cifra;
- Lo script si può compilare;
- Molti problemi anche di Teoria dei numeri si basano sul calcolo di un log.

```
/* log2(n) It works well with integer values n */
{log2(n) = local (k, i, nq, extrsup);
k=n;
i=0;
extrsup=0;
nq=0;
if(!ispower(n,2), nq = 1);
while(k!=0,
k=shift(k,-1);
i++;
);
/* if n isn't a power of 2 -> nq = 1 */
if( nq==1,
extrsup = i;
while( 2^extrsup > n,
extrsup=extrsup-0.00011;
);
i=extrsup+1;
);
return(i-1);
}
/*
** logb(n,b)
*/
{logb(n,b) = local (k, l, m);
k=log2(n);
l=log2(b);
m=1. * k/l;
return(m);
}
/*
** loge(n) (Nepero)
*/
{loge(n) = local (k, extrsup, m);
k=log2(n);
extrsup = 2; /* 2 ^ 2 = 4 */
while( 2^ extrsup > 2.71828182 ,
extrsup=extrsup-0.0001;
);
m = 1. * k/extrsup;
return(m);
}
```

Conclusioni

Questa piccola raccolta spero contribuisca allo studio e all'approfondimento dei metodi e degli algoritmi per il calcolo dei logaritmi.

Invito i lettori a porre domande, suggerimenti o a fornire, eventualmente, altri metodi noti o meno, per il calcolo del logaritmo.

Giovanni Di Maria
Gruppo Eratostene Caltanissetta