## Esercizio 2

“Si implementi un programma per il calcolo della costante del numero di Eulero-Mascheroni utilizzando vari metodi. Si discuta l’influenza dei vari tipi di errore di calcolo (algoritmici, analitici e di round off) “

**Introduzione**

La costante di Eulero - Mascheroni è una costante matematica, usata principalmente nella teoria dei numeri e nell'analisi matematica. È definita come limite della differenza tra la serie armonica troncata e il logaritmo naturale:

$$\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \int_1^n \frac{1}{x}dx\right) = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln n\right) = \lim_{n\to\infty}\left(H_n - \ln n\right)$$

ove $H_n$ è l’ennesimo numero armonico. E’ ancora ignoto se la costante gamma sia a tutti gli effetti un numero irrazionale. Con l'avvento della tecnologia si è riusciti a calcolare un numero sempre più alto di numeri decimali ma la dimostrazione della sua trascendenza non è stata ancora trovata.

Verrà utilizzata come riferimento nel programma una versione troncata del numero gamma pari a 0.577215664901532.

L’utilizzo di un calcolatore per la risoluzione di un problema matematico comporta che i dati sono sempre affetti da errori, per “costruzione” infatti un calcolatore numerico è in grado di rappresentare solo un numero finito di cifre. Inoltre, le operazioni elementari eseguite su tali numeri possono, a loro volta, produrre risultati non rappresentabili esattamente nel calcolatore.

Sui calcolatori si usa la rappresentazione floating-point normalizzata di Von Neumann $0.d_1d_2 \dots d_n * b^{\alpha}$ ove con d intendo la mantissa e b la base utilizzata (tipicamente la base 2) e α la caratteristica. lo standard definisce due tipi di numeri con differenti precisione:

float → Numeri in singola precisione, che utilizzano 32 bit di memoria.

double → Numeri in doppia precisione, che utilizzano 64 bit di memoria.

Si nota facilmente che c'è un errore intrinseco nella rappresentazione dei numeri dovuto alla finitezza della memoria utilizzata. Non tutti i numeri sono infatti rappresentabili, e il *gap* tra un numero e il successivo varia in base all'ordine di grandezza utilizzato.

Questi errori (di Round Off) sono inevitabili, dobbiamo quindi cercare di non farli prevalere nell’algoritmo (ad esempio sommando cifre di ordine di grandezza molto differente, il contributo del più piccolo si perderà nel processo di troncamento). In questo caso in cui viene trattato un numero irrazionale (o comunque supposto tale) ci sarà la presenza di questo tipo di errore, il numero ad infinite cifre decimali verrà quindi approssimato nei termini finiti che la memoria dell’elaboratore ci concede.

Un'ulteriore sorgente di errore è dovuta all'algoritmo utilizzato. Quando si valuta, ad esempio, il valore numerico di una serie convergente, non è possibile calcolare la somma di infiniti termini. La serie verrà approssimata con una somma parziale, arrestata a una certa quantità

prefissata. Algoritmi diversi, tuttavia, convergono con diverse velocità. L'errore di convergenza è dunque un errore puramente teorico, che deriva dal bisogno di interrompere la computazione a un determinato step. Si può tentare di ovviare a questo problema usando algoritmi che coinvolgono serie che convergono velocemente alla soluzione.

E' possibile citare tra le sorgenti di errori più frequenti anche quelle che coinvolgono all'interno di un algoritmo sottrazioni di numeri circa uguali e dello stesso ordine di grandezza (cancellazione catastrofica) che porteranno inevitabilmente ad errori.

**Risultati e discussione**

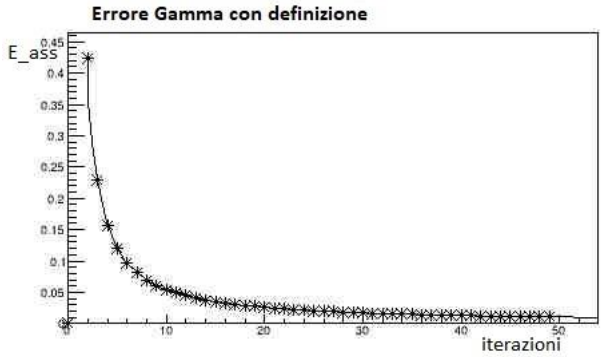
La Gamma di Eulero è stata calcolata in questo programma con 6 metodi differenti:

Metodo 1 – Definizione

Applicando la definizione

$$\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln n\right)$$
$$= \lim_{n\to\infty}(H_n - \ln n),$$

è stato possibile calcolare la gamma di Eulero-Mascheroni. Nel programma il numero di iterazioni aumenta per valutare l'aumento della precisione aumentando il numero delle somme parziali. Questo è il grafico dell'errore assoluto in funzione del numero di iterazioni:




Anche aumentando il numero di iterazioni l'andamento positivo di diminuzione dell'errore non cambia. La serie armonica diverge ma è approssimabile al logaritmo di k, questo fa si che sottraendolo con l'altro termine logaritmico si ottenga la gamma di Eulero-Mascheroni. L'errore in questo caso si riduce via via che aumentano le somme parziali, riducendone l'errore algebrico dovuto al trasformare una serie a termini infiniti in una a termini finiti.

Andamento simile lo possiamo osservare col metodo 2:

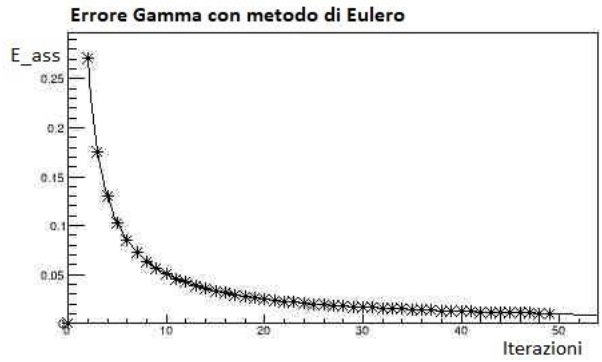
Metodo 2 – Metodo di Eulero

$$\gamma = \lim_{n\to\infty}\left[\sum_{k=1}^{n}\frac{1}{k} - \sum_{k=1}^{n}\ln\left(1+\frac{1}{k}\right)\right]$$
$$= \lim_{n\to\infty}\sum_{k=1}^{n}\left[\frac{1}{k} - \ln\left(1+\frac{1}{k}\right)\right]$$

Questa relazione è ottenuta rimaneggiando la definizione sostituendo il termine logaritmico con la seguente serie telescopica

$$\sum_{k=1}^{n} \ln\left(1+\frac{1}{k}\right)$$

L'andamento dell'errore ottenuto è il seguente:




Si può notare che l'andamento è simile, ma il termine calcolato converge più velocemente alla costante di Eulero-Mascheroni. L'errore algoritmico diminuisce in quanto diminuisce il numero delle somme parziali richieste (la convergenza è aumentata). Inoltre a differenza della definizione qui vengono calcolate le somme parziali e solo alla fine il limite, migliorando la stima della gamma.

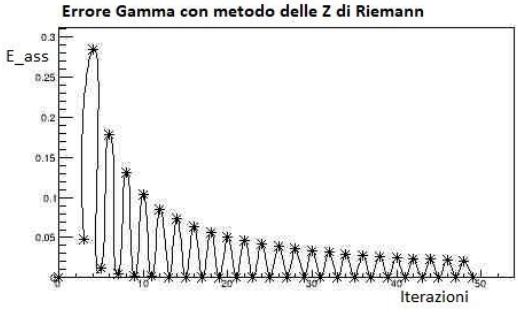
Per questi due metodi si può notare che anche aumentando il numero di iterazioni l'errore permarrà a causa della presenza degli errori di round off, la stima non aumenterà di precisione per il troncamento dei valori, aumentando di dimensione i termini i valori aggiuntivi incideranno progressivamente sempre meno fino a sparire.

Metodo 3 – Metodo delle Z di Riemann

Questo metodo sfrutta la seguente relazione con le Z di Riemann:

$$\gamma = \sum_{n=2}^{\infty} (-1)^n \frac{\zeta(n)}{n}$$
$$= \ln\left(\frac{4}{\pi}\right) + \sum_{n=1}^{\infty} \frac{(-1)^{n-1}\,\zeta(n+1)}{2^n\,(n+1)}$$

L'andamento dell'errore assoluto è il seguente:

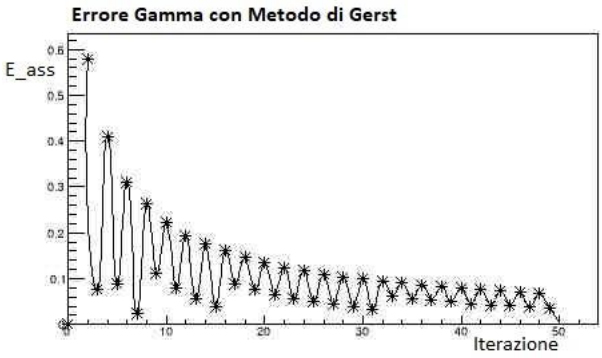


Simile è il metodo 4

Metodo 4 – Metodo di Gerst

$$\gamma = \sum_{n=1}^{\infty} (-1)^n \frac{\lfloor \lg n \rfloor}{n}$$

Ove la serie a segni alterni come per il metodo delle Z e a numeratore abbiamo la funzione parte intera del logaritmo. L'andamento dell'errore:



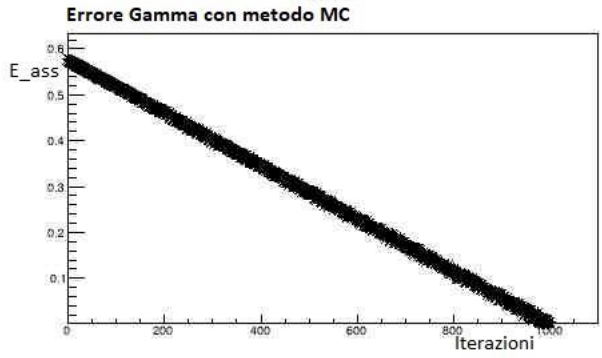

In entrambi i casi la precisione è sicuramente peggiore dei due metodi precedenti. Abbiamo sempre la presenza dell'errore algoritmico dovendo troncare una serie e gli errori dovuti alla finitezza dell'aritmetica di un elaboratore. Ma ad essi si aggiunge l'errore analitico dovuto alla sottrazione di termini che aumentando le iterazioni diventano simili. La convergenza del metodo di Gerst è più lenta rispetto a quello delle Z che risulta essere un metodo invece più preciso. Una parte dell'informazione nel metodo di Gerst viene inoltre persa attraverso la funzione parte intera che può portare ad errori nell'arrotondamento all'aumentare delle dimensioni dell'iterazione.

Metodo 5 – Integrale con Metodo MC

Posso passare al continuo e dalle serie ottenere la seguente relazione integrale:

$$\begin{aligned}
\gamma &= -\int_0^{\infty} e^{-x} \ln x \, dx \\
&= -\int_0^{1} \ln \ln \left(\frac{1}{x}\right) dx \\
&= \int_0^{\infty} \left(\frac{1}{1-e^{-x}} - \frac{1}{x}\right) e^{-x} \, dx \\
&= \int_0^{\infty} \frac{1}{x} \left(\frac{1}{1+x} - e^{-x}\right) dx
\end{aligned}$$

Nel programma verrà utilizzata la seconda relazione delle 4 presentate. Per calcolare l'integrale si è sfruttato un Metodo MC con numero di punti generati via via crescente ottenendo il seguento risultato:




Per aumentare la precisione della stima è necessario avere un elevato numero di punti e questo aumenta decisamente il costo computazionale dell'operazione. Il risultato però è estremamente positivo, l'errore algebrico è ridotto non avendo serie da interrompere e anche l'errore analitico ridotto avendo una somma continua di termini. Rimane il problema inevitabile degli errori di Round-off che non consentono di aumentare ulteriormente la precisione della stima ma come si può notare il risultato è comunque soddisfacente. Resta comunque il problema del costo computazionale, oltre ai numeri generati le operazioni logaritmo in C++ sono piuttosto dispendiose in termini di risorse macchina.

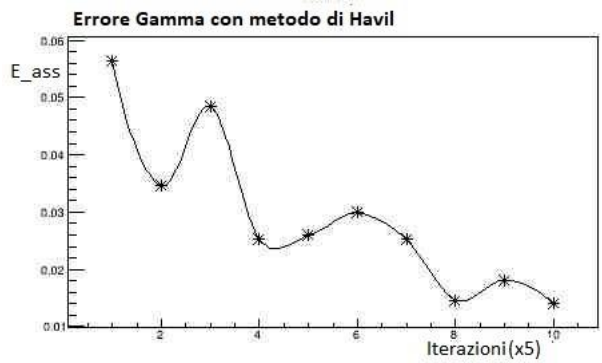
Metodo 6 – Metodo di Havil

I metodi usati fino ad adesso sono stati comunque positivi a livello d'elaborazione con errori e precisioni comunque tenuti sotto controllo anche aumentando il numero di iterazione. Problematico è invece il Metodo di Havil che porta ad un algoritmo instabile sebbene con un errore ridotto.

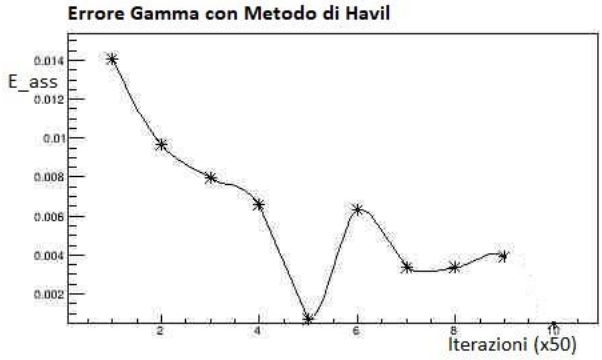
La relazione utilizzata è la seguente:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n-1} \left( \left\lceil \frac{n}{k} \right\rceil - \frac{n}{k} \right) = \gamma$$

con la mantissa della frazione n/k. Dalle considerazioni fatte in precedenza sulla mantissa e sulla finitezza dei numeri rappresentabili da un elaboratore già si può ipotizzare un problema di precisione:




Questo è il grafico con 50 iterazioni (totali). Se le iterazioni vengono aumentate ulteriormente però:




Si può notare subito come l'errore assoluto cali con l'aumentare delle iterazioni ma poi ricomincia ad aumentare. Questo è dovuto alla presenza di un errore analitico (sottrazione di termini simili) unita alle problematiche Round-Off della mantissa con l'aumentare delle iterazioni. L'algoritmo è efficiente con un numero ridotto di iterazioni, all'aumentare di esse la qualità della precisione peggiora.