**Pigreco in C++ (Metodo di Archimede)**

# Obiettivo

Trovare un valore di **pigreco** più preciso rispetto a quello che è possibile trovare utilizzando il foglio elettronico.

## Restrizioni

È possibile usare soltanto metodi e funzioni conosciuti al tempo di Archimede (no funzioni trigonometriche, no calcolo radice quadrata attraverso funzione predefinita del computer).

# Svolgimento

## Teoria (Metodo di Archimede e procedura misteriosa)

Il calcolo del pigreco è stato fatto utilizzando il “**Metodo di Archimede**” ossia una serie di divisioni successive tra perimetro e diametro di un poligono inscritto ad una circonferenza unitaria dove ad ogni iterazione il numero dei lati (all’inizio uguale a 6) raddoppia al fine di trovare un valore di pigreco sempre più preciso.

È necessario ricordare che il valore trovato di pigreco sarà sempre un’approssimazione dovuta a limiti tecnici del calcolatore che non è in grado di effettuare calcoli oltre una certa soglia di precisione.

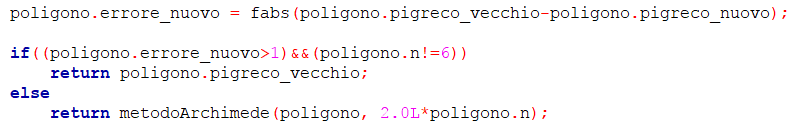
Bisogna quindi ricorrere ad una serie di operazioni che consentono di ricavare la misura del lato di un poligono avente numero doppio di lati (2n) rispetto al poligono precedente (n), ricordandosi che Archimede non aveva a disposizione la radice quadrata come calcolata dalle calcolatrici.

Per fare questo ho seguito le linee guida fornitemi creando una funzione che effettua la radice quadrata di un radicando passato come parametro utilizzando una procedura approfondita in classe, denominata procedura misteriosa, che consente di trovare un valore molto vicino alla radice del prodotto di due numeri dati grazie a una serie di frazioni ripetute. Inserendo quindi il numero 1.0 e il radicando questa procedura è in grado di calcolare la radice quadrata del radicando con buona approssimazione.

## C++ (main.cpp)

Per avere maggiore chiarezza e ordine all’interno del mio programma, ho ritenuto utile creare una struttura “Poligono” che racchiudesse in sé tutti gli attributi del poligono con il quale si andava ad operare (numero di lati, apotema, lato, perimetro, pigreco, ecc.) in modo da poterne avere accesso utilizzando una sola variabile di tipo Poligono.

La funzione che calcola il pigreco acquisisce come parametro un Poligono e il numero iniziale di lati ritornando poi, dopo varie iterazioni, il valore del pigreco. È una funzione ricorsiva la quale, dopo aver calcolato i vari attributi del Poligono di *n* lati (compreso il pigreco), richiama se stessa raddoppiando di volta in volta il numero dei lati del poligono (1° iterazione = 6, 2° iterazione = 12, 3° iterazione = 24 ..) solo se il controllo di fine ciclo non è soddisfatto.

La soluzione che ho trovato più funzionale per concludere il ciclo è quella di controllare se il valore della differenza in valore assoluto tra il pigreco rilevato in un’iterazione e quello rilevato nell’iterazione ad essa precedente sia maggiore di 1 in un’iterazione diversa dalla prima (poiché questa differenza nella prima iterazione è necessariamente maggiore di 1 essendo una differenza tra il primo pigreco trovato, 3, e lo 0). Questo perché, *inserendo altre condizioni*, ho notato che, dopo aver calcolato il “pigreco massimo” correttamente, *la funzione non si fermava* ma continuava facendo assumere al pigreco valori assurdi e privi da ogni logica passando per esempio da 3.14…… a -1E-1 a +6E-2 per poi interrompersi ritornando all’esterno uno di questi valori sbagliati.

Condizione di fine ricorsività funzione

Per ovviare a questo problema ho pensato di far continuare il ciclo fino a quando il computer non fa succedere qualcosa (probabilmente a causa della sua limitata capacità di calcolo) che stravolge tutti i risultati ottenuti come ad esempio una differenza che passa da essere nell’ordine di grandezza di 1E-18 a 1.

Ho elaborato il mio codice in modo da renderlo compatibile con due tipi di valori (\_\_float128, long double) ed è possibile selezionare il tipo con cui si vuole far eseguire il programma modificando il codice sorgente main.cpp alla riga 10 scrivendo:

* *#define FLOAT128* (se si vuole calcolare il pigreco utilizzando il tipo **\_\_float128**)
* *#define LONGDOUBLE* (se si vuole calcolare il pigreco utilizzando il tipo **long double**)

# RISULTATI

In questa tabella sono riportati i valori e la precisione di pigreco trovati utilizzando ciascun tipo:

|  |  |  |
| --- | --- | --- |
| **TIPO** | **VALORE PIGRECO** | **ERRORE** |
| \_\_float128 | 3.14159265358979323… | 5E-18 |
| long double | 3.14159265358958236… | 1E-14 |

I risultati denotano che:

* il pigreco ottenuto con **\_\_float128** è il più preciso avendo un errore di *5E-18*.
* il pigreco ottenuto con **long double** ha un errore “alto” considerando che questo tipo dovrebbe garantire una precisione maggiore.

Effettuando tentativi utilizzando la radice quadrata implementata in automatico dal computer (*sqrt()*, libreria math.h), il pigreco ricavato utilizzando **long double** mostrava un errore di *1E-15*. Questo mi fa pensare che tutti gli errori registrati durante questo progetto siano da ricondurre alla difficoltà del calcolatore di operare a questo livello di approssimazione, accentuate dalle operazioni effettuate che, utilizzando un metodo approssimato per ricavare la radice quadrata di un numero, non possono garantire una precisione e affidabilità tale da non aspettarsi errori.

Utilizzando **\_\_float128** posso comunque essere soddisfatto del risultato in quanto il pigreco finale trovato è abbastanza preciso da “*vincere”* la precisione del pigreco ottenuta attraverso il foglio di calcolo.

**PIGRECO FINALE:** 3.14159265358979323..

**ERRORE:** 5E-18

**TIPO VARIABILI:** \_\_float128