DOCUMENTAZIONE PROGETTO PROGRAMMAZIONE E CALCOLO SCIENTIFICO

RAFFINAMENTO MESH TRIANGOLARE: metodo complesso

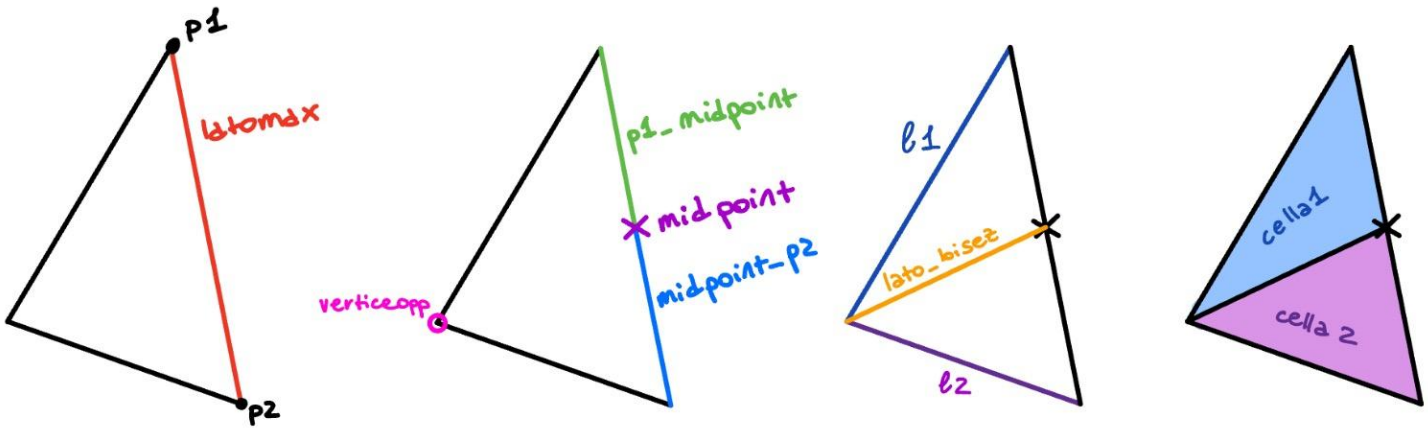
Lorenzo de Gregorio, Andrea Grasso, Luca Mercuriali

Descrizione:

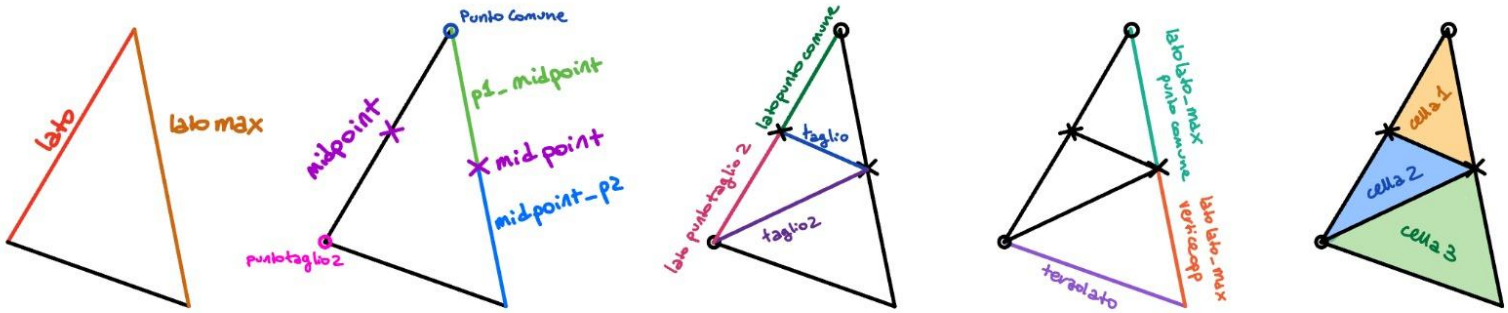
L'obiettivo del raffinamento è quello di generare una versione più dettagliata di una mesh triangolare bidimensionale, a patto che questa sia ammissibile. Perché una mesh sia considerata ammissibile, deve rispettare il seguente criterio: per ogni cella della mesh, le sue celle adiacenti possono condividere un lato completo o solo un vertice. Non è permesso che più di un lato completo o più di un vertice siano adiacenti a una cella.

L'algoritmo deve trovare il lato maggiore e dividerlo a metà. si inizia selezionando un lato qualsiasi del triangolo:

- Se il lato selezionato è il lato maggiore divido la cella in 2. Dichiaro l1 il lato che congiunge punto 1 al vertice opposto ed l2 il lato che congiunge il punto 2 al vertice opposto. Creo 2 celle e riempio la matrice di adiacenza delle due celle.



- Se il lato selezionato non è il lato maggiore divido la cella in tre parti:
  Prendo il punto medio sia del lato maggiore che del lato selezionato. Poi trovo il punto comune tra i due lati e il vertice opposto del lato maggiore.
  Eseguo poi due tagli che partono entrambi dal punto medio del lato maggiore e uno va a puntotaglio2 e l'altro a pm del lato selezionato. Chiamo poi i lati opportunamente per poi dare un ordine preciso alle celle che creo. Avendo diviso il triangolo in tre parti devo creare le tre nuove celle generate dalla separazione e riempire quindi l'adiacenza opportunamente.



STRUTTURA LOGICA

Il programma è diviso in 5 file :

- "main_test": C++ Source file (.cpp) - contiene le istruzioni per runnare tutti i test;
- "main_program": C++ Source file (.cpp) - istruzioni per importare i dati dai file dati e dare inizio al processo di raffinamento della mesh;
- "empty_class": C++ Header file (.hpp) - Dichiarazione delle classi, costruttori e metodi/funzioni degli oggetti.
- "empty_class": C++ Source file (.cpp) - contiene l'implementazione dei costruttori, dei metodi e delle funzioni dichiarate in "empty_class.hpp";
- "test_empty": C++ Header file (.hpp) – implementazione dei test;

La struttura dei cinque file è la seguente

MAIN_PROGRAM.cpp

Dà inizio all'import dei dati creando tre vettori di oggetti, uno di tipo Point, uno di tipo Segment uno di tipo Cell, riempiti chiamando le funzioni void d'import, tre metodi che prendono in input il percorso del file d'interesse e il rispettivo vettore da riempire passato per referenza usando un oggetto di tipo istringstream.

1. Criteri d'arresto:
   Metto un numero massimo d'iterazioni max_it, e una tolleranza sull'area theta
2. Apro i tre file mettendoli come punti, lati e triangoli;
   Inizializzo delle variabili che sono l'id massimo che sto usando,
   in modo tale che so qual è l'id più piccolo che devo prendere successivamente
   (ne creo uno per i punti, uno per i lati e uno per i triangoli)
3. Stampo l'area della cella maggiore e il numero di celle prima del raffinamento.
4. Faccio il raffinamento usando MaxCelle, LatoMaggiore e RaffinaCella

EMPTY_CLASS.hpp

1. Dichiaro le classi:
    a. Dichiaro la classe PUNTO in cui inizializzo come double l'ascissa e l'ordinata
    b. Dichiaro la classe SEGMENTO in cui inizializzo due punti, la distanza tra di essi (lunghezza) due puntatori (uno dal primo punto al punto medio e uno dal secondo punto al punto medio) e due celle (la prima e la seconda cella adiacenti, quelle più vicine); poi calcolo la lunghezza del segmento con il teorema di Pitagora
    c. Dichiaro la classe CELLA in cui inizializzo i vettori dei punti e dei lati, l'area come double e un fattore booleano che uso per distruggere le celle

2. Chiamo la funzione Cell: aggiungo i tre punti al vettore punti, aggiungo i tre lati al vettore lati,
   Pongo l'indicatore booleano = vero e calcolo l'area del triangolo

3. METODI: operazioni con il lato più lungo della cella
a. Dichiaro il metodo per trovare LATO_MAGGIORE:

i. se il lato 0 è più grande del lato 1 e se il lato 1 è più lungo del lato 2→ latomax (lato_maggiore)= lato 0
ii. altrimenti se il lato 1 è maggiore del lato 2 → latomax=lato 1
iii. altrimenti (→lato 2 più grande di lato 0 e di lato 1)→ latomax=lato2

b. dichiaro metodo per trovare il PUNTO_MEDIO:
calcolo il punto medio su x e il punto medio su y e aggiungo il punto medio al vettore dei punti

c. dichiaro metodo per trovare VERTICE_OPPOSTO:
prendo in input la cella e il lato, inizializzo a 0 un contatore i e prendo come punto il punto della cella con quel contatore. uso ciclo while (→ finché il punto = al primo punto del lato oppure il punto = al secondo punto del lato (ovvero se io prendo un vertice e quel vertice è uno dei due vertici del lato selezionato)) aumento di 1 il contatore e pongo di nuovo il punto come punto della cella con quel contatore.

d. dichiaro metodo per trovare VERTICE_COMUNE:
prendo in input i primi due lati e inizializzo come intero un punto (vertice comune).

i. se il primo punto del lato 1 = al primo punto del lato 2 → il vertice comune è il punto 1.
ii. se il primo punto del lato 1 = secondo punto del lato 2 → il vertice comune è il punto 1.
iii. altrimenti il punto comune è il punto 2.

4. Importo dai file csv i vettori di punti, segmenti e celle
5. Dichiaro i seguenti METODI sulla cella:
   a. Dichiaro il metodo DIVIDI_CELLA:
   divide la cella dimezzando il lato, se poi il lato non è il lato maggiore divide ancora creando un lato tra il punto medio di lato e del alto maggiore
   b. Dichiaro il metodo RAFFINA_CELLA:
   passo come argomenti una cella e un lato. seleziono la cella adiacente alla cella selezionata usando il lato u comune per decidere quale cella scegliere. Chiamo poi la funzione DividiCella. Devo dividere in due la mia funzione. Se il lato massimo della cella successiva coincide con il lato massimo della cella selezionata, devo dividere questa cella successiva usando DividiCella. Se invece il lato comune tra le due celle non è il lato massimo per la cella successiva allora devo raffinare questa cella successiva (chiamando quindi la funzione RaffinaCella)

EMPTY_CLASS.cpp

1. Dichiaro le funzioni per importare gli oggetti:
    a. dichiaro la funzione per importare i PUNTI: devo leggere il file dividendo i vari dati con spazi o punti e virgola e poi aggiungo il punto al vettore dei punti
    b. dichiaro la funzione per importare i SEGMENTI: devo leggere il file dividendo i vari dati con spazi o punti e virgola e poi aggiungo il segmento al vettore dei segmenti
    c. dichiaro la funzione per importare le CELLE: devo leggere il file dividendo i vari dati con spazi o punti e virgola; costruisco le adiacenze (se il lato 1,2,3 ha cella1 o cella2= - 1 metto l'id di cella in cella1 o cella2)

2. dichiaro la funzione per DIVIDERE_CELLA:

a. Prima di tutto controllo se il lato è già stato diviso uccido la cella, altrimenti la divido.
    ➔ Se il puntatore dal primo vertice al punto medio oppure se il puntatore dal secondo vertice al punto medio o se il puntatore al punto medio sono minori di zero
    ➔ creo il punto medio, aggiungo 1 al massimo ID dei punti e creo nuovi lati, che saranno quelli dal primo vertice al punto medio e dal secondo vertice al punto medio;
    poi aggiungo 2 al contatore dei segmenti.
b. Dichiaro come intero il vertice opposto usando la funzione VerticeOpposto. Questa funzione agisce sulla base della distinzione in due casi illustrata precedentemente: uno è il caso in cui il lato selezionato sia il lato maggiore del triangolo, l'altro è il caso in cui non sia il lato maggiore.

3. dichiaro la funzione RAFFINA_CELLA:
   inizializzo lato maggiore con funzione LatoMaggiore, inizializzo la cella successiva e gli assegno il valore corretto.

4. dichiaro la funzione MAX_CELLE:
   inizializzo un contatore per la size dei triangoli (celle),
   pongo max=-1 e l'area massima =0,
   se il booleano=true e se l'area del triangolo è maggiore dell'attuale area massima
   allora assegna all'area massima l'area di quel triangolo (tutto in un ciclo for) .

Scelta della struttura dati e degli algoritmi e import dei dati

I dati della mesh sono forniti da tre file .csv in input contenenti rispettivamente la tabella dei punti, la tabella dei lati e la tabella delle celle.

- Di ogni punto è fornito il Marker (intero positivo), l'Id (intero positivo univoco all'interno dei punti), l'ascissa X (numero reale), l'ordinata Y (numero reale).
- Di ogni lato è fornito il Marker (intero positivo), l'Id (intero positivo univoco all'interno dei segmenti), l'Id del punto di origine del lato (pensato come vettore), l'Id del punto di fine del lato (pensato come vettore).
- Di ogni cella è fornito l'Id (intero positivo univoco all'interno delle celle), gli Id dei tre vertici della cella, gli Id dei tre lati della cella.

La classe Segment ha come attributi gli Id dei suoi due punti estremi, del suo punto medio (inizializzato al valore -1 che significa che il lato non è stato ancora diviso), gli Id dei suoi due mezzi lati (inizializzati al valore -1, che significa che il lato non è stato ancora diviso), gli Id delle due celle adiacenti per quel lato (inizializzati al valore -1).

La classe Cell ha come attributi un vettore contenente i tre Id dei suoi vertici, un vettore contenente i tre Id dei suoi lati e un Booleano (flag) che indica se la cella è stata già raffinata o non ancora.

Una volta importati i dati della mesh di partenza, memorizziamo per ogni classe il primo valore Id non ancora utilizzato, sfruttando il fatto che gli Id in input vanno ordinatamente da 0 al numero totale degli elementi -1.

Una volta costruita una cella, si rintracciano i suoi lati grazie all'Id, in modo da agevolare la ricerca delle celle adiacenti con un costo computazionale O(1) di accesso a un vettore per posizione.

Abbiamo salvato gli oggetti in un vettore std::vector<Object>. L'utilizzo di un vettore ha il vantaggio di permettere l'accesso a un suo elemento in O(1) se si conosce la sua posizione, che può essere uguale al suo Id se si usa un vettore non ordinato. Di contro, i vettori sfruttano un'allocazione della memoria variabile, attraverso l'utilizzo dei puntatori, talvolta poco affidabili.

Abbiamo scartato l'idea di utilizzare un array in quanto risulta difficile prevedere quanta memoria allocare per l'array, e si può ovviare al problema dei puntatori

utilizzando la posizione dell'elemento nel vector, sfruttando il fatto che questa corrisponde esattamente all'Id univoco. Abbiamo anche scartato l'uso di una mappa in quanto l'area delle celle non è una potenziale chiave univoca.

Abbiamo deciso di calcolare il massimo dell'area delle celle esistenti a ogni iterazione al posto di ordinare inizialmente le celle per area e mantenere in seguito un inserimento ordinato, anche se ciò comporta un costo computazionale più alto.

Il costruttore degli oggetti è direttamente nell'argomento del push_back perché utilizzato dentro a delle funzioni; Creiamo quindi una variabile globale permanente. Se si chiamasse il costruttore all'interno della funzione creerebbe una variabile locale che smetterebbe di esistere una volta terminata la chiamata della funzione, finendo per perdere quello che andiamo ad aggiungere al vettore tramite push_back.

Analisi del costo computazionale e dell'utilizzo della memoria

Il costo computazionale del programma è dato fondamentalmente dalla ricerca della cella di area massima tra tutte le celle attualmente esistenti, che si ripete a ogni iterazione. Per k = numero di raffinamenti (iterazioni), il costo dell'algoritmo è:

$$\sum_{i=0}^{k} o\,(n_i)$$

(n_i numero di celle alla i-1 iterazione)

RISULTATI:

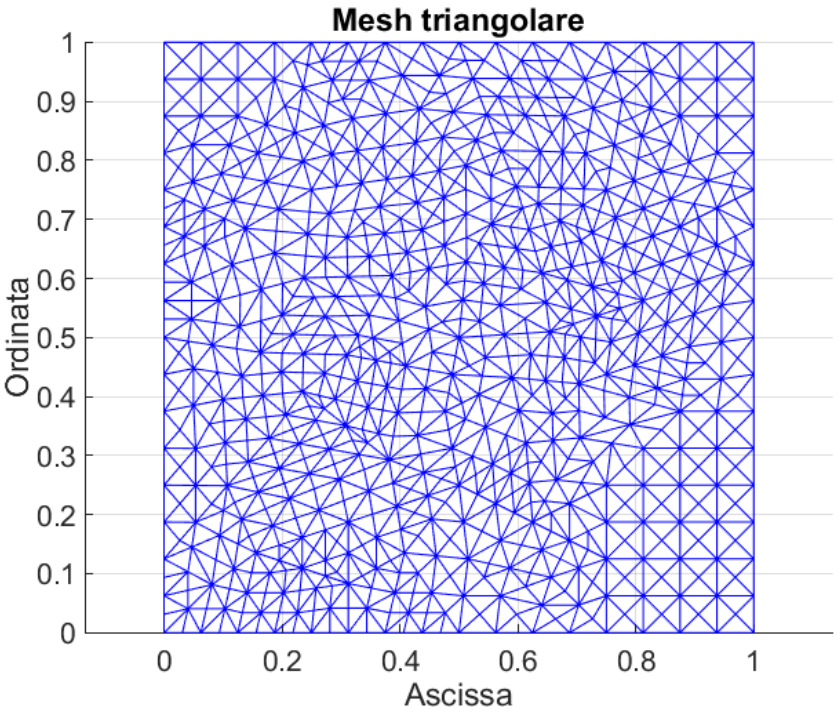
nel main_program.cpp abbiamo imposto due criteri di arresto. un numero di interazioni massime, max_it, e un valore theta di tolleranza sull'area. modificando questi due parametri otteniamo mesh sempre più fini (aumentando max_it e diminuendo theta) oppure più grossolane (diminuendo max_it e aumentando theta).
Qui sotto mostriamo delle immagini delle mesh finali in base a questi due criteri d'arresto

Theta=10^-3

max_it=800

```
raffinamento_program
Area Cella Maggiore (Prima di Dividere): 0.00989101
Numero Celle (Prima di Dividere): 144
Area Cella Maggiore (Dopo Divisione): 0.000997347

tolleranza (theta = 0.001) raggiunta in 664 iterazioni
Tempo Impiegato: 0.039197 s
15:10:02: C:\Users\hp\Desktop\project_PCS\Progetto_PCS\
```

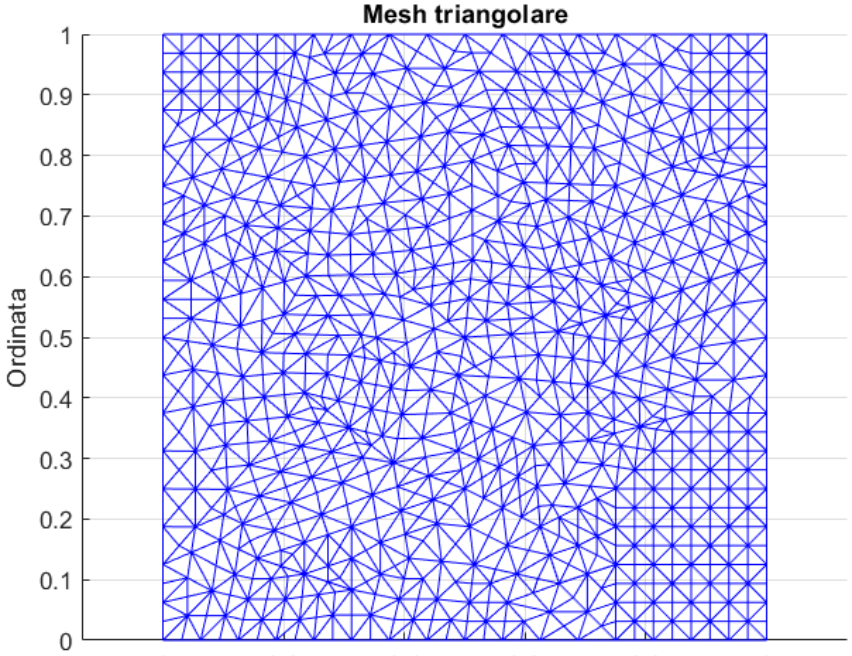


Theta=10^-4

max_it=800

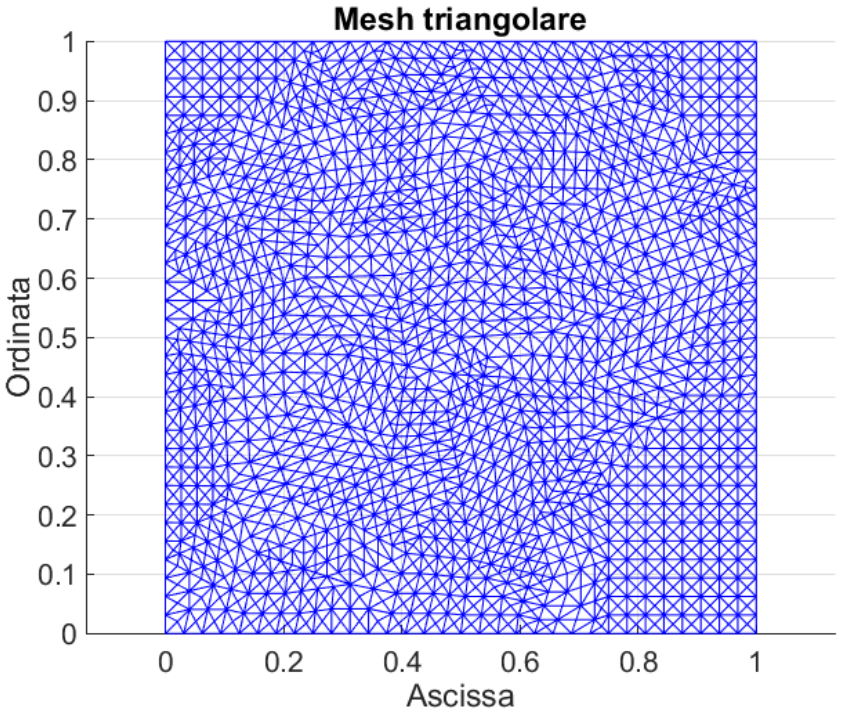
```
raffinamento_program
12:14:51: Starting C:\Users\hp\Desktop\project_PCS\
Area Cella Maggiore (Prima di Dividere): 0.00989101
Numero celle (Prima di Dividere): 144
Area Cella Maggiore (Dopo Divisione): 0.000969008

raggiunto limite massimo di iterazioni (800),
12:14:55: C:\Users\hp\Desktop\project_PCS\Progetto_
```



Theta=10^-4

max_it=2000



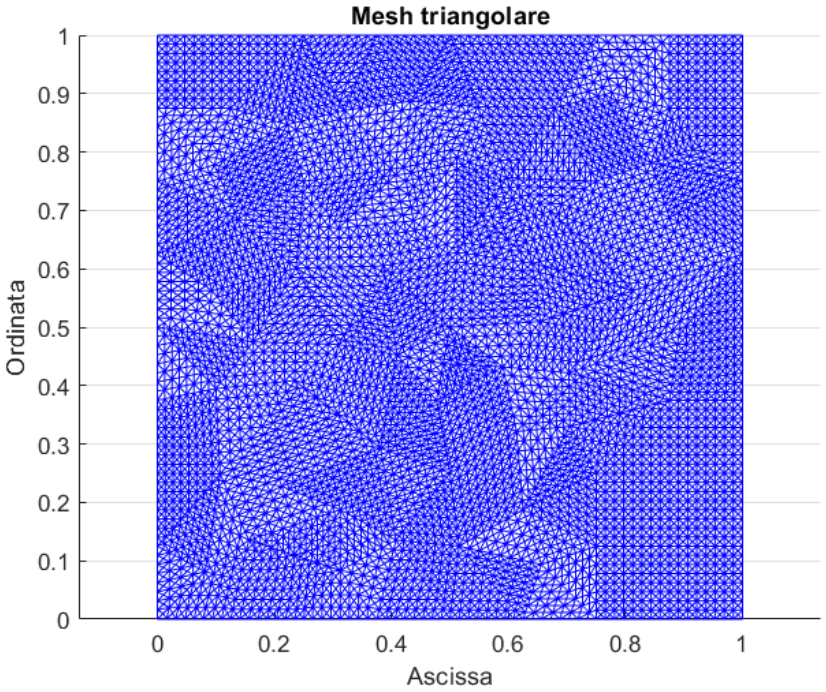
raffinamento_program

```
12:16:42: Starting C:\Users\hp\Desktop\project_PCS\
Area Cella Maggiore (Prima di Dividere): 0.00989101
Numero celle (Prima di Dividere): 144
Area Cella Maggiore (Dopo Divisione): 0.000366211

raggiunto limite massimo di iterazioni (2000),
12:16:44: C:\Users\hp\Desktop\project_PCS\Progetto_
```

Theta=10^-4

max_it=9000



raffinamento_program

```
15:00:04: Starting C:\Users\hp\Desktop\project_PCS\Progetto
Area Cella Maggiore (Prima di Dividere): 0.00989101
Numero celle (Prima di Dividere): 144
Area Cella Maggiore (Dopo Divisione): 9.87851e-05

tolleranza (theta = 0.0001) raggiunta in 7679 iterazioni
15:00:08: C:\Users\hp\Desktop\project_PCS\Progetto_PCS\Proj
```