Progetto Laboratorio di Calcolo

Calcolo dell'integrale definito mediante il metodo Monte Carlo

Iacobucci Luca(20035727)

Fasano Lorenzo(20028396)

Raffaele Alessandro(20025449)

Traccia progetto

"Scrivere un programma per valutare mediante la generazione di numeri casuali (vedere dispense cap. 9) l'integrale definito di una funzione su un intervallo [a,b] specificato dall'utilizzatore (ma comunque non più ampio di 0 < x < 10): determinare il rettangolo con base sull'asse x che contiene tutto il grafico della funzione per a < x < b, generare punti distribuiti a caso nel rettangolo e calcolare il valore approssimato dell'integrale definito con il metodo descritto nelle dispense.
Per stimare l'errore di calcolo, eseguire il programma aumentando il numero di punti utilizzati".

Introduzione

Il programma creato ha lo scopo di calcolare approssimativamente il valore dell'integrale definito di una funzione su un intervallo [a,b] specificato dall'utente. La funzione integranda utilizzata è `sqrt(abs(sin(x)))`, che è definita per tutti i valori di x. Il metodo utilizzato è il metodo Monte Carlo, una tecnica statistica che utilizza numeri casuali per risolvere problemi matematici.

Metodo

Il metodo Monte Carlo consiste nel generare punti casuali all'interno di un rettangolo che contiene il grafico della funzione nell'intervallo [a,b]. Il numero di punti che cadono sotto il grafico della funzione viene utilizzato per stimare l'integrale definito.

La formula utilizzata per calcolare l'integrale approssimato è la seguente:

$$Integrale \approx (b - a) * \left(\frac{punti\ sotto\ il\ grafico}{numero\ totale\ di\ punti}\right)$$

Codice

```c
/*
AUTORI
Iacobucci Luca      20035727
Fasano Lorenzo      20028396
Raffaele Alessandro 20025449
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
```

```c
// Funzione da integrare: sqrt(abs(sin(x)))
double integrand(double x) {
   return sqrt(fabs(sin(x)));
}

// Funzione per generare un numero casuale in [min, max]
double random_double(double min, double max) {
   return min + (max - min) * ((double)rand() / RAND_MAX);
}

int main() {
   double a, b;           // Intervallo [a, b]
   int n_points;          // Numero di punti casuali
   char output_file[100]; // Nome file di output

   // Input: intervallo [a, b], numero di punti e nome file
   printf("Inserisci l'estremo inferiore dell'intervallo a (0 <= a < 10): ");
   scanf("%lf", &a);
   printf("Inserisci l'estremo superiore dell'intervallo b (a < b <= 10): ");
   scanf("%lf", &b);

   if (a <= 0 || b > 10 || a >= b) {
       printf("Intervallo non valido.\n");
       return 1;
   }

   printf("Inserisci il numero di punti casuali da generare: ");
   scanf("%d", &n_points);

   if (n_points <= 0) {
       printf("Il numero di punti deve essere maggiore di 0.\n");
       return 1;
   }

   printf("Inserisci il nome del file di output per i punti (es. data_1000.txt): ");
   scanf("%s", output_file);

   // Determinazione del rettangolo di bounding
   // l'incremento di 0.001 è scelto per bilanciare la necessità di precisione con
l'efficienza computazionale.
   double max_f = 0;
   for (double x = a; x <= b; x += 0.001) {
       double f_x = integrand(x);
       if (f_x > max_f) {
           max_f = f_x;
       }
```

```
    }

    // Apertura file per salvare i punti
    FILE *file = fopen(output_file, "w");
    if (!file) {
        printf("Errore nell'apertura del file %s\n", output_file);
        return 1;
    }



    /*
    Generazione punti casuali e calcolo dell'integrale
    Questa parte del codice genera punti casuali nell'area del rettangolo di bounding e
conta
    quanti di questi punti cadono sotto la curva della funzione integrand.
    Questi punti vengono anche salvati in un file di output per eventuali analisi
successive.
    */
    int inside = 0;
    srand(time(NULL)); // Inizializzazione del generatore di numeri casuali

    for (int i = 0; i < n_points; i++) {
        double x = random_double(a, b);
        double y = random_double(0, max_f);

        if (y <= integrand(x)) {
            inside++;
        }

        // Salva i punti nel file
        fprintf(file, "%f %f\n", x, y);
    }

    fclose(file);

    // Area del rettangolo
    double rectangle_area = (b - a) * max_f;

    // Valore approssimato dell'integrale
    double integral = rectangle_area * ((double)inside / n_points);

    printf("\nValore approssimato dell'integrale: %.6f\n", integral);
    printf("Numero di punti interni: %d\n", inside);
    printf("Errore stimato rispetto al valore noto (7.47626): %.6f\n", fabs(integral -
7.47626));

    printf("Punti salvati nel file: %s\n", output_file);
```

```
    return 0;
}
```

Grafici

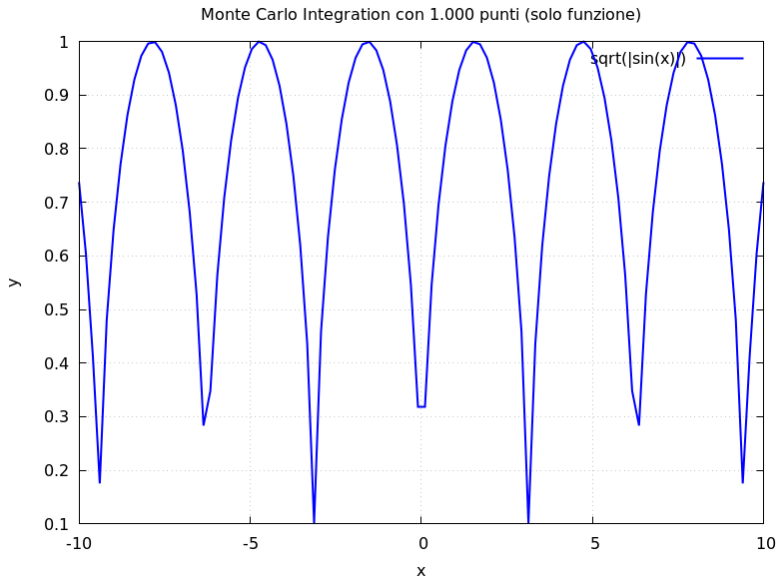
Per generare i grafici, è stato utilizzato il software gnuplot.
Gli script per creare i grafici sono:

Per “data_1000.txt”:

```
set terminal pngcairo size 800,600;
set output 'integrale_1000.png';

set title "1.000 punti";
set xlabel "x";
set ylabel "y";
set grid;

# Plotta la funzione teorica e i punti generati
plot sqrt(abs(sin(x))) with lines lw 2 linecolor rgb "blue" title "sqrt(|sin(x)|)", \
'data_1000.txt' using 1:2 with points pointtype 7 linecolor rgb "red" title "Punti Monte Carlo";
```

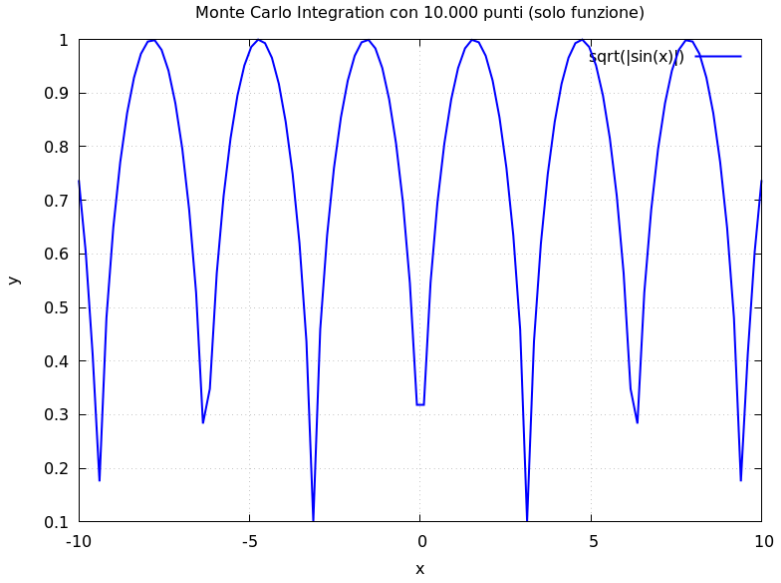


Per “data_10000.txt”:

```
set terminal pngcairo size 800,600
set output 'integrale_10000.png'

set title "Monte Carlo Integration con 10.000 punti (solo funzione)"
set xlabel "x"
set ylabel "y"
set grid
```

```
# Plotta la funzione teorica
plot sqrt(abs(sin(x))) with lines lw 2 linecolor rgb "blue" title
"sqrt(|sin(x)|)"
```

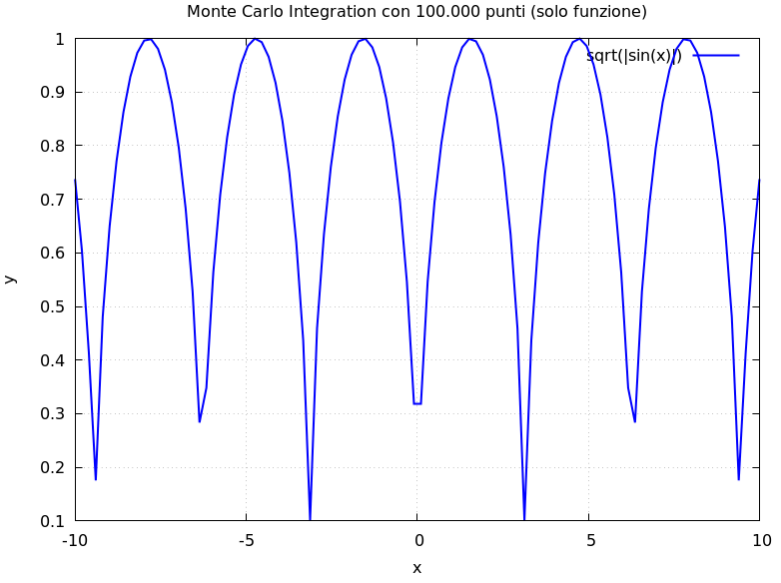


Per “data_100000.txt”:

```
set terminal pngcairo size 800,600
set output 'integrale_100000.png'

set title "Monte Carlo Integration con 100.000 punti (solo funzione)"
set xlabel "x"
set ylabel "y"
set grid

# Plotta la funzione teorica
plot sqrt(abs(sin(x))) with lines lw 2 linecolor rgb "blue" title
"sqrt(|sin(x)|)"
```



Risultati

Per stimare l'errore di calcolo, il programma è stato eseguito aumentando il numero di punti utilizzati. I risultati ottenuti sono stati confrontati con il valore approssimato dell'integrale definito da 0 a 10, che è circa 7.47626.

Con 1.000 punti:

```
Inserisci l'estremo inferiore dell'intervallo a (0 < a < 10): 0
Inserisci l'estremo superiore dell'intervallo b (a < b <= 10): 10
Inserisci il numero di punti casuali da generare: 1000
Inserisci il nome del file di output per i punti (es. data_1000.txt): data_1000.txt

Valore approssimato dell'integrale: 7.300000
Numero di punti interni: 730
Errore stimato rispetto al valore noto (7.47626): 0.176260
Punti salvati nel file: data_1000.txt
```

Con 10.000 punti:

```
Inserisci l'estremo inferiore dell'intervallo a (0 < a < 10): 0
Inserisci l'estremo superiore dell'intervallo b (a < b <= 10): 10
Inserisci il numero di punti casuali da generare: 10000
Inserisci il nome del file di output per i punti (es. data_1000.txt): data_10000.txt

Valore approssimato dell'integrale: 7.507000
Numero di punti interni: 7507
Errore stimato rispetto al valore noto (7.47626): 0.030740
Punti salvati nel file: data_10000.txt
```

Con 100.000 punti:

```
Inserisci l'estremo inferiore dell'intervallo a (0 < a < 10): 0
Inserisci l'estremo superiore dell'intervallo b (a < b <= 10): 10
Inserisci il numero di punti casuali da generare: 100000
Inserisci il nome del file di output per i punti (es. data_1000.txt): data_100000.txt

Valore approssimato dell'integrale: 7.474400
Numero di punti interni: 74744
Errore stimato rispetto al valore noto (7.47626): 0.001860
Punti salvati nel file: data_100000.txt
```

Conclusioni

Il metodo Monte Carlo si è dimostrato efficace nel calcolare l'integrale definito della funzione `sqrt(abs(sin(x)))` su un intervallo specificato dall'utente. Aumentando il numero di punti utilizzati, l'errore di calcolo si riduce, avvicinandosi sempre più al valore reale dell'integrale.