APPROSSIMAZIONE NUMERO DI NEPERO METODO MONTE CARLO

Relazione progetto:

Metodi Matematici e Statistici (6 CFU)

GIOELE CAGEGGI

I*ndice:*

INTRODUZIONE

In matematica il numero e è una costante il cui valore è approssimativamente

$$e \approx 2.7182818284\ldots$$

È la base della funzione esponenziale e^x e del logaritmo naturale.

Può essere definita in vari modi, il più comune tra i quali è il limite della successione

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n;$$

Oltre alle rappresentazioni analitiche esatte per calcolare e esistono altri metodi *stocastici* per stimarlo.

Uno di questi è il Metodo Monte Carlo HIT-or-MISS che fa uso di variabili aleatorie artificiali (cioe' generate da un calcolatore in maniera randomizzata) per la risoluzione di problemi matematici.

In tale progetto è stato analizzata un'altra soluzione alternativa, sempre stocastica, ideata da K. G. Russell noto russo, che ne propose il calcolo del numero di Nepero come esercizio "banale", di cui ne analizzeremo il funzionamento analitico e pratico ed infine confronteremo i risultati di entrambi i metodi per vedere qual è il più efficiente

FUNZIONAMENTO TEORICO ALGORITMO ALTERNATIVO RUSSO

Si parte da una sequenza finita di variabili causali indipendeti

X1 , X2 , … , Xn.

distribuite uniformemente nell'intervallo [0,1].

Definito K come il numero di somme parziali n_i di variabili *Xn* tali che esse siano strettamente minori di 1 si ha:

$$V = \frac{\sum_{i=0}^{K} n_i}{K};$$

dove

$$n_i = min(n : X_1 + X_2 + \ldots + X_n > 1);$$

Allora il valore atteso E[V] coincide con *e*.

ES:
K=2 -> 2 TRIAL
TRIAL 1 -> Viene rendirizzato e memorizzato un valore pari a 3 in n1
Random1: 0.0180 X1
Random2: 0.4596 X2
Random3: 0.7920 X3

TRIAL 1 -> Viene rendirizzato e memorizzato un valore pari a 2 in n2
Random1: 0.0480 X1
Random3: 0.8030 X2

V = (n1+n2) / 2 = 2,5

Per un numero di TRIAL sufficientemente grande si prova che E[V] è pari ad *e*

FUNZIONAMENTO PRATICO ALGORITMO ALTERNATIVO STOCASTICO

Il tutto inizia mediante un main dove si distinguono tre parti:

- IMPOSTAZIONI:

```
/****Impostazioni****/
cout.precision(10);
srand(time(NULL));
```

Qui viene impostato fix di visione di cifre pari a 10 nello standard output, inoltre con srand(time(NULL)) si setta il seed per il pseudo generatore di numeri casuali che la funzione di libreria <time.h> rand() userà.

- DATI:

```
/****Data****/
int eventi;
double nepero;
double e = 2.718281828;
```

Qui vengono istanziate ed eventualemente inizializzate le variabili che verranno usate durante l'esecuzione della simulazione

- LOOP:

```
/****Loop****/
while(true){

    /****Interfaccia****/
    cout << "Calcola il numero di NEPERO 'e' mediante il metodo Stocastico"<<endl;
    cout << "Inserisci il numero di eventi con cui calcolare il valore di NEPERO [100, 10000000]"<<endl<<"> ";
    cin  >>  eventi;
    cout <<  endl ;

    /****Calcola****/
    nepero = CalcolaNepero(eventi);

    /****Mostra Risultato****/
    cout << "Il valore di NEPERO stimato e': " << nepero << " e differenzia da e di :" << nepero-e;
    cout << endl << endl;

}
```

Il loop si suddivide ulteriormente 3 sezioni:

- La prima *[INTERFACCIA]* rappresenta l'interfaccia che permette al programma di presentarsi e chiedere all'utente di inserire il numero di eventi per il quale si vuole stimare il valore di Nepero mediante la simulazione.
- La terza *[CALCOLA]* rappresenta l'area di Working dove viene richiamata la funzione di calcolo che approfondiremo tra non molto
- La quarta *[MOSTRA RISULTATO]* rappresenta l'interfaccia dei risultati dove vengono mostrati: il numero di nepero stimato, la sua differenza con 'e' già fisso a 9 cifre di approssimazione.

Analizziamo la sezione [CALCOLA]:

Viene richiamata la funzione CalcolaNepero() che prende in input il numero di eventi che l'utente ha inserito nell'interfaccia e che intende usare il per iniziare la simulazione

```
/****Algoritmo che calcola il valore di 'e' ****/
double CalcolaNepero(int N_eventi){

 /****Inizializzazione variabili****/
 double Somma = 0, n_Somme = 0, n = 0, nepero;
 int trial = N_eventi;

 /****Loop****/
 while( trial > 0){

   /****Inserisce in Somma un valore random [0,1]****/
   Somma += GeneraRandom();
   /*****Rappresenta Ni*****/
   n_Somme++;
   /****Check****/
   if(Somma>1){
   //n=numero renderizzato di ogni trial per arrivare ad un somma>1
       n += n_Somme;
       /*****Reset&DecrementoTrial****/
       n_Somme = 0;
       Somma = 0;
       trial--;
   }
 }
 /****Calcola ****/
  nepero = CalcolaRapporto( n , N_eventi) ;
  return nepero;
}
```

La funzione prevede di inizializzare variabili necessarie.
Si entra quindi in un loop che si ripeterà esattamente N_eventi-esime volte: all'interno della variabile somma aggiunto per ogni trial un valore random (0,1) mediante la seguente funzione.

```
double GeneraRandom () {
   double random_number=((double)rand()/(RAND_MAX));
   return random_number;
}
```

Viene, inoltre, tenuto conto del numero di cicli necessari alla trial per arrivare ad avere la variabile Somma>1 nella variabile n_Somma.

Se la variabile Somma è maggiore di 1 allora viene memorizzato il numero di cicli effettuati per arrivare a cio all'interno della variabile n, successivamente avviene un reset delle variabili per la prossima trial.

Adesso mediante la funzione CalcolaRapporto viene calcolato il rapporto: $V = \frac{\sum_{i=0}^{K} n_i}{K}$;
Dove K=N_eventi e la sommatoria è n.
Il valore di ritorno di tale macro funzione è l'approssimazione del numero di Nepero.

METODO MONTE CARLO HIT-or-MISS

Con il termine di Metodo Monte Carlo oppure metodo MC, vengono in generale denominate tutte quelle tecniche che fanno uso di variabile aleatorie artificiali (cioe' generate al calcolatore) per la risoluzione di problemi matematici.

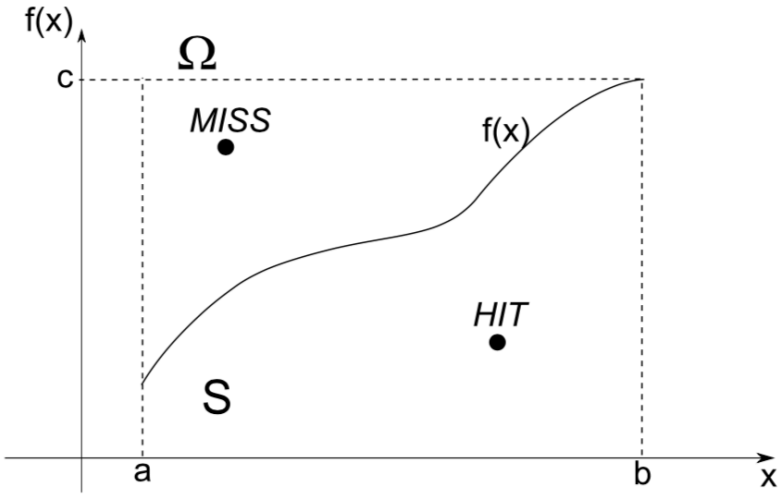
Una di queste è la tecnica HIT-or-MISS:

Data una funzione f(x) definita in un intervallo [a,b] e limitata

cioè $\exists\ c \in R$:

$$0 \leq f(x) \leq c$$

Consideriamo il rettangolo



Se adesso generiamo N vettori casuali (x_i,y_i) i = 1,2,...,N la probabilità p che il vettore (x,y) cada sotto la curva f(x) e'

$$p = \frac{area\ S}{area\ \Omega} = \frac{\int_a^b f(x)dx}{c(b-a)} = \frac{I}{c(b-a)}$$

L’integrale I si può stimare nel seguente modo

$$I = \int_a^b f(x)dx \ \sim \ \theta_1 = c(b-a)\frac{N_H}{N}$$

Dove N_H = numero hits, cioè il n° di volte in cui il punto (x_i,y_i) cade sotto la curva e N = numero totale di numeri generati

L’applicazione di questo metodo non e’ ristretta solamente ai problemi di natura statistica, come forse si potrebbe pensare dato l’utilizzo di distribuzioni di probabilita’, ma include tutti quei casi in cui si riesce a trovare un collegamento tra il problema in esame ed il comportamento di un certo sistema aleatorio.

Sicuramente questo non e’ il modo piu’ efficiente per ricavare tale costante, in quanto la procedura del campionamento simulato porta ad un risultato che e’ sempre affetto da imprecisione e approssimazione.

Tuttavia grazie all’avento di calcolatori sempre più avanzati è possibile utilizzare tale metodo avendo risultati soddisfacenti, effettuando un numero di calcoli elevatissimi, in tempi relativamente brevi.

FUNZIONAMENTO TEORICO
ALGORITMO MONTE CARLO HIT-or-MISS

1. Generare una successione di 2N numeri casuali compresi tra 0 ed 1

$$\{U_i\}\ i = 1, 2, \ldots, N \qquad U_i \in [0,1]$$

2. Disporre i numeri casuali in N coppie

$$(U_1,U_1'),(U_2,U_2'),\ldots,(U_N,U_N')$$

in modo che ogni numero casuale $\{U_i\}$ è usato una sola volta.

3. Calcolare

$$x_i = a + U_i(b - a) \text{ e } f(x_i) \qquad i=1,2,\ldots,N$$

4. Contare il n° di volte N_H in cui

$$f(x_i) \geq cU_i'$$

5. Stimare l'integrale I con

$$\theta_1 = c(b-a)\frac{N_H}{N}$$

6. Il valore stimato rappresenta la nostra approssimazione di ***e***

Da come si evince nella sezione *risultati e confronti*, l'algoritmo HIT-or-MISS risulta di gran lunga migliore nell'approssimazione del valori di nepero rispetto al metodo stocastico russo;

Per tanto viene analizzato in fondo tale algoritmo per cui è stato messa in evidenza :

- *Intervallo di confidenza* con livello di di fiducia alpha=0,03

  Intervallo di valori che ha una determinata probabilita' di contenere il parametro oggetto di stima [valore atteso]. Alpha rappresenta il grado di attendibilita' del nostro intervallo

- Calcolo della *varianza* e della *deviazione standard*

  Varianza: Indice che valuta quanto i dati siano concentrati intorno al valore medio dei dati, minore è il valore maggiore sarà tale concentrazione e viceversa.

  Deviazione Standard: Fortemente legata alla varianza [di cui ne è la radice quadrata] ed indica la dispersione dei dati intorno ad un indice di tendenza

- Calcolo della varianza e della deviazione standard usando la tecnicha di riduzione della varianza basata sulla variabile antitetica

  Tecnica Antitetica: Si prova a determinare due stimatori unbiased dell'integrale I, tale che abbiano una forte correlazione negativa.

FUNZIONAMENTO PRATICO ALGORITMO METODO MONTE CARLO HIT-or-MIS

Il tutto inizia mediante un main dove si distinguono tre parti:

- IMPOSTAZIONI:

```
/****Impostazioni****/
cout.precision(10);
srand(time(NULL));
```

Qui viene impostato fix di visione di cifre pari a 10 nello standard output, inoltre con srand(time(NULL)) si setta il seed per il pseudo generatore di numeri casuali che la funzione di libreria <time.h> rand() userà.

- DATI:

```
/****Data****/
double nepero, eventi;
double AS, N_hit=0, varianza=0, devstd=0,devstdAntitetica=0, varianzaAntitetica=0, e = 2.718281828;
double I1=0, I2=0;
double somma;
```

Qui vengono istanziate ed eventualemente inizializzate le variabili che verranno usate durante l'esecuzione della simulazione

- LOOP:

```
 /****LoopProgramma****/
 while(true){

   /****Reset Dati****/
     nepero=0;
     eventi=0;

     /****Interfaccia****/
     cout << "Calcola il numero di NEPERO 'e' mediante il metodo Monte Carlo HIT-or-MISS"<<endl;
     cout << "Inserisci il numero di eventi con cui calcolare il valore di NEPERO [100, 10000000]"<<endl<<"> ";
     cin  >>  eventi;
     cout <<  endl ;

     /****Calcola****/
     AS     = CalcolaAS(eventi, &N_hit);
     nepero = CalcolaNepero(AS);
     CalcolaIntervalloConfidenza(AS, eventi, nepero, &I1, &I2);
     varianza = CalcolaVarianza(AS, N_hit, somma);
     devstd = CalcolaDevSTD(varianza);
     varianzaAntitetica = CalcolaVarianzaAntitetica(AS, eventi, somma);
     devstdAntitetica = CalcolaDevSTDAntitetica(varianzaAntitetica);


     /****MostraRisultati****/
     cout <<"/****RISULTATI****/"<<endl;
     cout << "Il valore di NEPERO stimato e': " << nepero <<endl;
     cout << "Differenza da 'e' di :" << nepero-e<<endl;
     cout << "L'intervallo di confidenza con livello di fiducia 0,03(99,7perc) e' : " << endl <<"[ " <<  I1 << " , " << I2 << " ]"<<endl;
     cout << "La varianza stimata e': " << varianza << endl;
     cout << "La Deviazione Standard stimata e': " << devstd << endl;
     cout << "La varianza ridotta mediante Variabile Antitetica e': " << varianzaAntitetica << endl;
     cout << "La Deviazione Standard Ridotta stimata e': " << devstdAntitetica << endl;
     cout <<"/****RISULTATI****/"<<endl;
     cout << endl << endl;

 }

return 0;
}
```

Il loop a sua volta si suddivide in 4 sezioni:

- La prima *[RESET DATI]* serve a resettare le variabili *nepero e eventi* per evitare conflitti ad ogni loop.
- La seconda *[INTERFACCIA]* rappresenta l'interfaccia che permette al programma di presentarsi e chiedere all'utente di inserire il numero di eventi per il quale si vuole stimare il valore di Nepero mediante la simulazione.
- La terza *[CALCOLA]* rappresenta l'area di Working dove vengono effettuati i vari calcoli che approfondiremo tra non molto.
- La quarta *[MOSTRA RISULTATI]* rappresenta l'interfaccia dei risultati dove vengono mostrati: il numero di nepero stimato, la sua differenza con 'e' già fisso a 9 cifre di approssimazione, l'intervallo di confidenza, la varianza e la deviazione standard, la varianza e deviazione standard con l'utilizzo della tecnica di riduzione della varianza mediante le variabili antitetiche.

Analizziamo la sezione [CALCOLA]:

```
/****Calcola****/
AS      = CalcolaAS(eventi, &N_hit);
nepero = CalcolaNepero(AS);
CalcolaIntervalloConfidenza(AS, eventi, nepero, &I1, &I2);
varianza = CalcolaVarianza(AS, N_hit, somma);
devstd = CalcolaDevSTD(varianza);
varianzaAntitetica = CalcolaVarianzaAntitetica(AS, eventi, somma);
devstdAntitetica = CalcolaDevSTDAntitetica(varianzaAntitetica);
```

Calcola AS:

```
/****Calcola il valore NH / N_eventi****/
double CalcolaAS(double N_EVENTI, double *N_hit){

  double trial=N_EVENTI;
  double x , y , AS , hit=0;

  while(trial>0){

    x=GeneraRandom();
    y=GeneraRandom();

    if( ControllaValoreSottoFX(x,y) ) {
       hit+=1;
    }
    trial--;
  }

  AS = CalcolaRapporto(hit, N_EVENTI);
  *N_hit=hit;

  return AS;
}
```

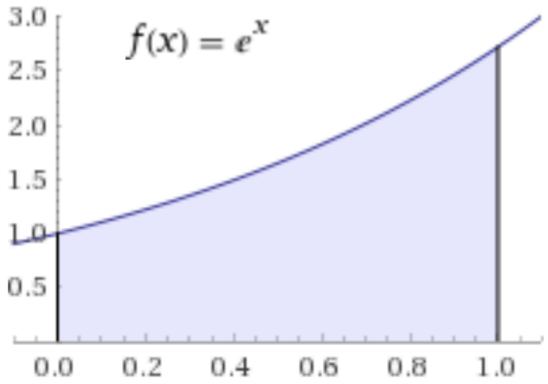
Essa permette di calcolare il rapporto tra il numero di HIT nell'***AreaSottesa*** e il numero totale di eventi, infatti prende come imput una variabile *eventi* e una variabile di memorizzazione *N_hit*.

Inizializzate alcune variabili essenziali si entra in un ciclo che dura esattamente N_EVENTI volte ad ogni ciclo viene generato un vettore [X,Y] mediante la funzione GeneraRandom() (implementata da me) che
Che prevede di generare valori random compresi tra 0 ed 1, esclusi, con distribuizione binomiale.

```
double GeneraRandom () {
   double random_number=((double)rand()/(RAND_MAX));
   return random_number;
 }
```

```
bool ControllaValoreSottoFX( double X ,  double Y){
    return exp(X)>=Y*2.71;
}
```

Succesivamente si effettua un controllo per verificare se il nostro vettore ricade sotto la curva della nostra funzione esponenziale e^x mediante ControllaValoreSottoFX(X,Y)



Tale immagine rappresenta la nostra funzione esponenziale nell'intervallo compreso tra [0,1] lungo l'asse x e [0,3] lungo l'asse y.

Se il nostro vettore [X,Y] generato in maniera randomizzata ricade all'interno allora si verifica un HIT e quindi la variabile che tiene conto del numero di HIT viene incrementata di una unità poichè la funzione booleana ControllaValoreSottoFX ritornerà true, cioè vero.

A tale punto si arriva alla conclusione del ciclo che prevede di decrementare la variabile trial, che ci permetterà di uscire del ciclo alla N_EVENTI-esima iterazione.

```
/****Calcola il rapporto tra due valori*****/
double CalcolaRapporto(double HIT, double N_EVENTI){
    return = (HIT / N_EVENTI);
}
```

Verra adesso calcolato il rapporto tra il numero di HIT e il numero totale di eventi N_EVENTI mediante la funzione CalcolaRapporto(hit, N_EVENTI) che non rappresenta ancora il nostro valore di Nepero ma bensi un estimatore, ricordiamo infatti che lavoriamo all'interno di un'area limitata che non rappresenta totalmente la nostra funzione FX che sappiamo essere infinita.

Dopo aver memorizzati quanti HIT sono capitati per N_EVENTI all'interno della variabile di memorizzazione N_hit la macro-funzione CalcolaAS termina ritornando al main il valore AS.

Calcola Nepero:

```
/**** (yMAX(b-a))*(N_HIT/N_EVENTI) ****/
double CalcolaNepero(double AS){
  return AS * 2.71 + 1;
 }
```

Essa ci permette di fixare, cioè sistemare, il valore stimato precedentemente AS per ottenere cosi il valore stimato di Nepero mediante la seguente formula

$$\theta_1 = c(b-a)\frac{N_H}{N}$$

el nostro caso Nh/N = AS, C=2,71 cioè la nostra Y_MAX e il valore +1 per fixare il risultato finale.
Il valore finale sarà una stima del numero di Nepero, certamente più è fixato il valore AS, mediante un numero di eventi elevemato, maggiore e migliore sarà l'approssimazione di tale valore.

Calcola Intervallo di Confidenza:

Tale funzione ci permette di stimare un intervallo di confidenza

```
/****Calcola l'intervallo di confidenza con alpha pari a 0,03[97%]****/
void CalcolaIntervalloConfidenza(double teta, double N_eventi, double nepero, double *I1, double *I2){
  double sigma = sqrt( teta * ( 1.0 - teta ) / N_eventi );
  *I1 = nepero-3*sigma;
  *I2 = nepero+3*sigma;
}
```

Prende in input parecchi parametri, tra cui il valore stimato AS =teta e due variabili di memorizzazione I1 e I2 che rappresentano rispettivamente il sinistra e il destra del nostro intervallo. Per prima cosi si determina la deviazione standard sigma mediante la formula accanto dove in questo caso teta = AS e N=N_eventi. Successivamente vengono calcolati I1 e I2 come mostrato, considerando un valore di affidabilità del 97%

$$\sigma = \sqrt{\frac{AS(1-AS)}{N}}$$

Calcola Varianza:

```
/****Calcola la varianza, un indice di variabilità****/
double CalcolaVarianza(double AS, double N_hit, double *somma){
  *somma += AS;
  double teta = *somma / N_hit;
  double teta2 = pow(*somma,2) / N_hit;

  return teta2 - pow(teta,2);
}
```

Calcolo in teta il rapporto tra la sommatoria di ogni stima AS all'interno di somma e il numero di N_hit affinche AS<minore di 1. Effettuo la stessa cosa anche per teta2 ma il numeratore è elevato al quadrato. Cio che ritorno è la differenza tra il valore già calcolato al quadrato teta2 - il quadrato di teta

$$< \theta > = \frac{\sum_{i=1} \theta_i}{N} \qquad < \theta^2 > = \frac{\sum_{i=1} \theta^2{}_i}{N}$$

$$\sigma^2 = < \theta^2 > - < \theta >^2$$

Calcola Deviazione Standard

```
/****Calcola La Deviazione standard tramite varianza****/
double CalcolaDevSTD(double varianza){
  return sqrt(varianza);
}
```

Calcola e ritorna al main la radice quadrata della varianza passata come parametro

Calcola Varianza Antitetica:

```
/****Calcola la varianza antitetica, un indice di variabilità ottimizzato****/
double calcolavarianzaantitetica(double N_EVENTI){
  double trial=N_EVENTI;
  double x , y , AS , hit=0;
  double xa, ya, ASa, hita=0;
  while(trial>0){

    x=GeneraRandom();
    y=GeneraRandom();
    xa=1-x;
    ya=1-y
    if( ControllaValoreSottoFX(x,y) ) {
       hit+=1;
    }
    if( ControllaValoreSottoFX(xa,ya) ) {
       hita+=1;
    }

    trial--;
  }

  AS = CalcolaRapporto(hit, N_EVENTI);
  ASa= CalcolaRapporto(hita, N_EVENTI);


  double tetaAntitetica = ( AS + ASa ) / 2;
  double tetaAntiteticaQuadro = pow(tetaAntitetica, 2);

  return tetaAntiteticaQuadro - pow(tetaAntitetica,2);
}
```

Il procedimento è molto simile all'algoritmo CalcolaAS tuttavia in questo caso viene il tutto raddoppiato poichè entrano in gioco le variabili antitetiche [Xa, Ya] calcolate a partire dalle variabili [X,Y] cioè

Successivamente viene calcolata la varianza come differenza tra

$$U_i[X,Y] \rightarrow \ \theta_i \ \ ; \ 1 - U_i[X,Y] \rightarrow \tilde{\theta}_i$$

$$\theta^A = \frac{\theta_i + \tilde{\theta}_i}{2}$$

$$\sigma^2 = <(\theta^A)^2> - <\theta^A>^2$$

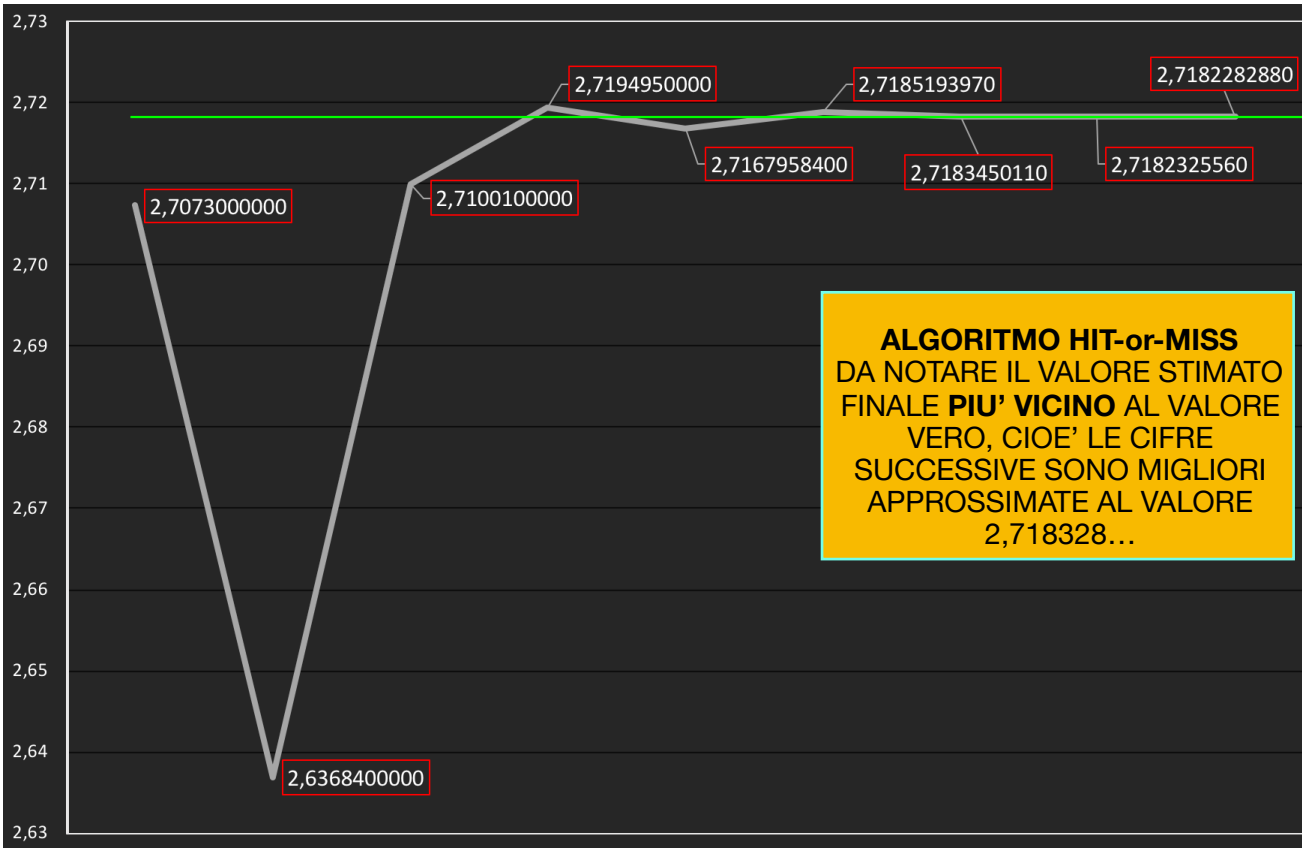
Calcola Deviazione Standard Antitetica:

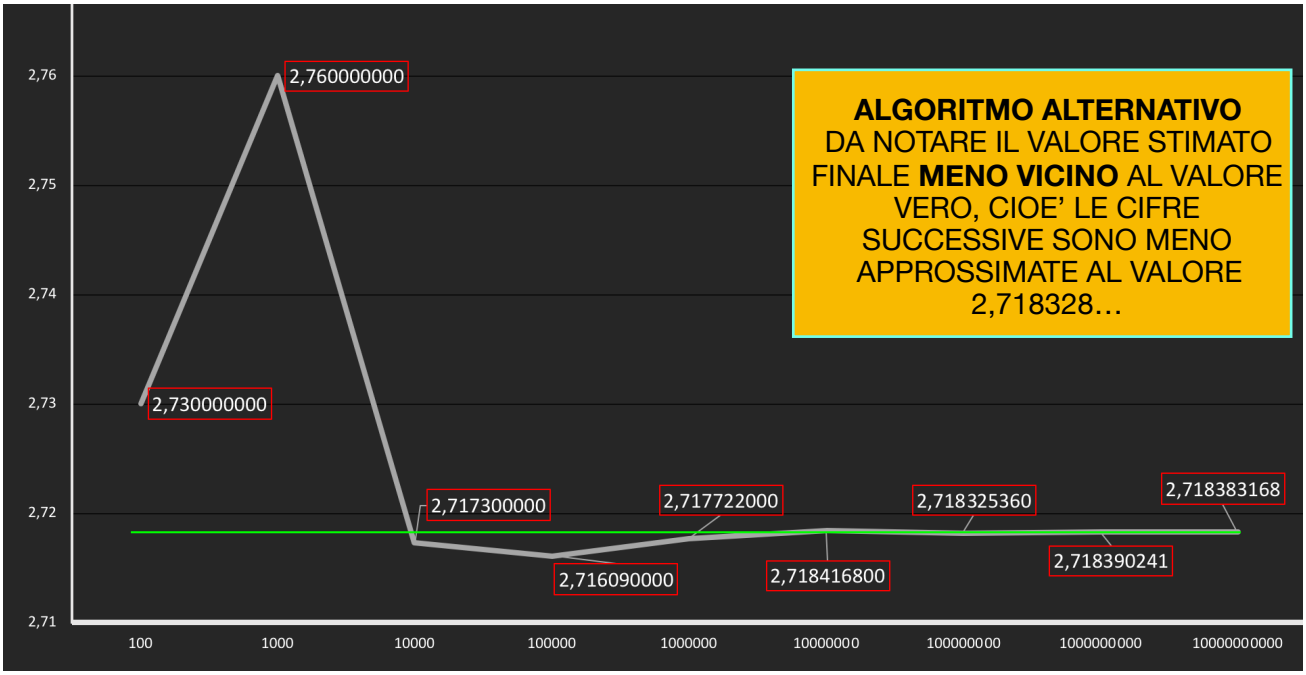
```
/****Calcola La Deviazione standard con varianza Antitetica****/
double CalcolaDevSTDAntitetica(double varianzaAntitetica){
   return sqrt(varianzaAntitetica);
}
```

Così come la Calcola Deviazione Standard anche questa semplicemente calcola la radice quadrata della varianza antitetica, quindi varianza ridotta, e la ritorna come output

RISULTATI E CONCLUSIONE

Confronto stima di e

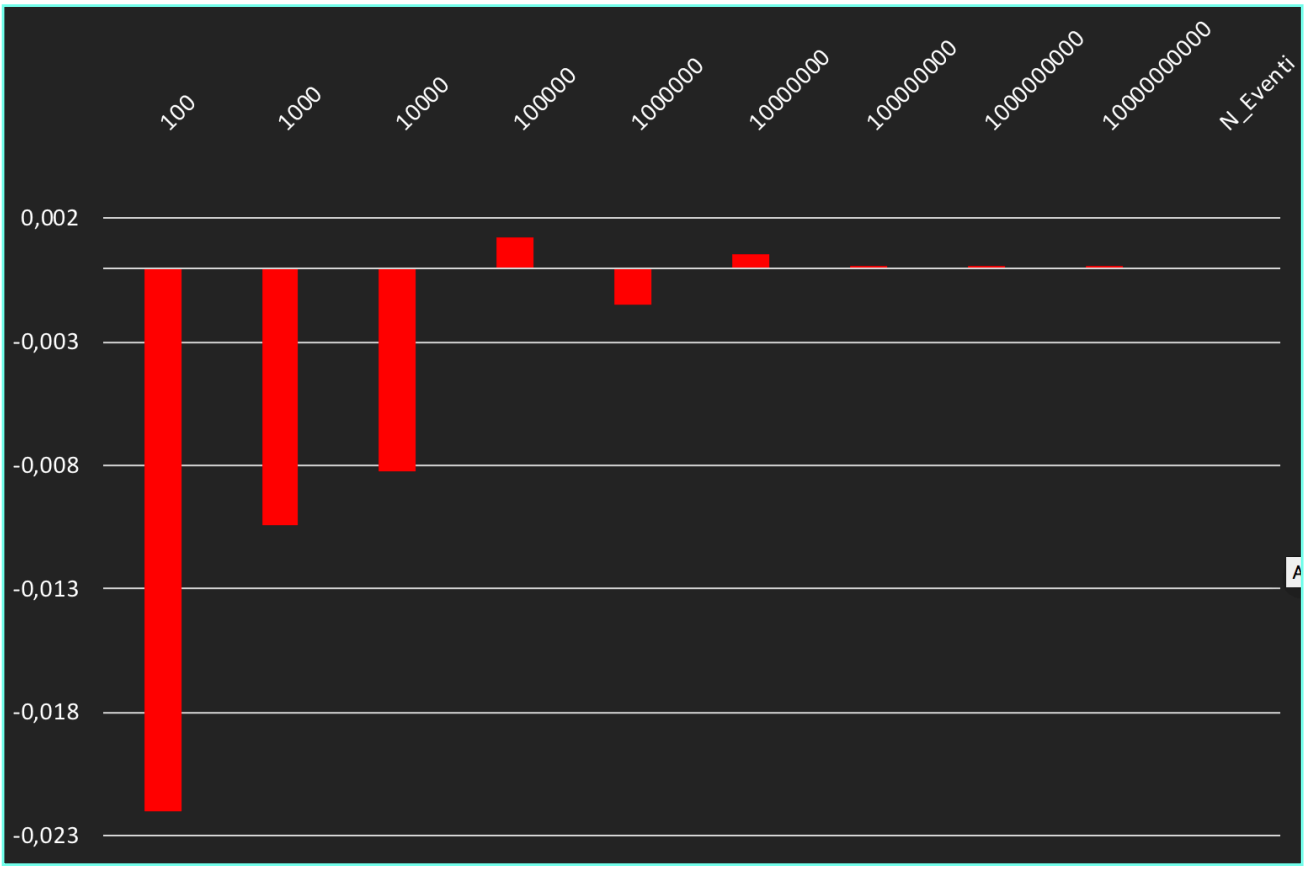




SOTTRAZIONE TRA

e-STIMATO -[MENO] *e-REALE*

METODO MONTE CARLO

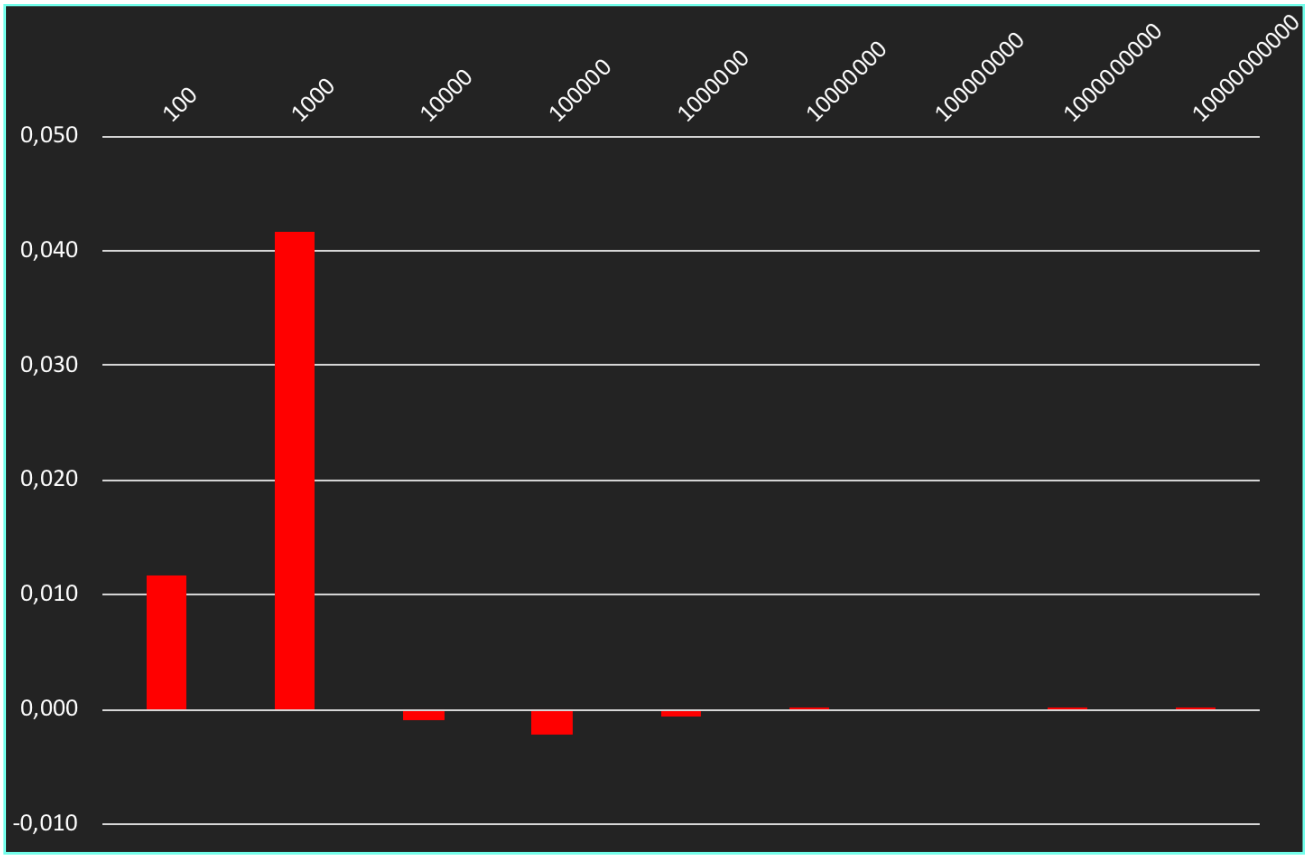


Confronto effettuato usando l'algoritmo **Monte Carlo HIT-or-MISS**
Il grafico mostra la differenza tra il valore 'e' *reale* a 9 cifre e il valore 'e' stimato tramite tale algoritmo.
Lungo l'asse delle y si indica la quantità di differenza tra il valore reale e il valore stimato, lungo x il numero di eventi per cui è stata fatta stima.
Si nota come inizialmente vi è una grande differenza, infatti è di circa -0,023[cioè il valore stimato è più piccolo del valore reale di 'e'] per una simulazione a 100 prove, per poi ridursi molto presto all'aumentare del numero di prove.
Da questo notiamo come il risultato dell'algoritmo è molto stabile per tanto anche l'algoritmo in se è stabile

SOTTRAZIONE TRA

e-STIMATO -[MENO] *e-REALE*

ALGORITMO ALTERNATIVO



Confronto effetuato usando l'**algoritmo alternativo stocastico.**
Il grafico mostra la differenza tra il valore 'e' *reale* a 9 cifre e il valore 'e' stimato tramite tale algoritmo.
Lungo l'asse delle y si indica la quantità di differenza tra il valore reale e il valore stimato, lungo x il numero di eventi per cui è stata fatta stima.
Si nota come inizialmente vi è una grande differenza, infatti è di circa 0,012[cioè il valore stimato è più grande del valore reale di 'e'] per una simulazione a 100 prove, per poi ridursi notevolmente e poi aumentare di nuovo.
Per tanto deduciamo chededuciamo che l'algoritmo produce un risultato instabile

CONCLUSIONE 1

Da questi due confronti comprendiamo subito che l'algoritmo alternativo stocastico risulta essere ,si una soluzione al problema ma, poco efficiente confrontato al metodo Monte Carlo HIT-or-MISS.

Infatti guardando i grafici a pagina 15 notiamo che entrambi inizialmente hanno valori molto lontani dal valore effettivo [negativo HIT-or-MISS, positivo Alternativo] ma con l'aumentare esponenziale del numero di eventi usati per realizzare la simulazione l'algoritmo basato sul Metodo Monte Carlo HIT-or-MISS produce un risultato più vicino al valore reale del numero di Nepero

A confermare questa ipotesi i grafici a pagina 16 [metodo Monte Carlo HIT-or-MISS] e 17 [metodo alternativo] mostrano la differenza effettiva tra il valore e *approssimato mediante la simulazione e il valore* e *"reale" .*

Da notare anche l'instabilità nel grafico a pagina 17 [metodo alternativo] rispetto al grafico a pagina 16 [metodo Monte Carlo HIT-or-MISS] .
Infatti vi è una tendenza a 0 lineare come differenza per HIT-or-MISS mentre vi è una tendenza a 0 altalenante come differenza per Metodo Alternativo.

Inoltre i tempi di esecuzione sono simili per entrambi nelle prime prove ma avanzando definitivamente no, infatti considerando solo ed esclusivamente il tempo necessario a stimare il valore di Nepero l'algoritmo HIT-or-MISS impiega circa 0:58.07 secondi mentre l'altro algoritmo cioè quello alternativo impiega circa 1:17.6 secondi, quasi 20 secondi di differenza nel caso in cui si fa stima con 10 miliardi di eventi.

Quindi possiamo affermare che l'algoritmo basato sul Metodo Monte Carlo HIT-or-MISS per la simulazione del numero di Nepero è nettamente migliore.

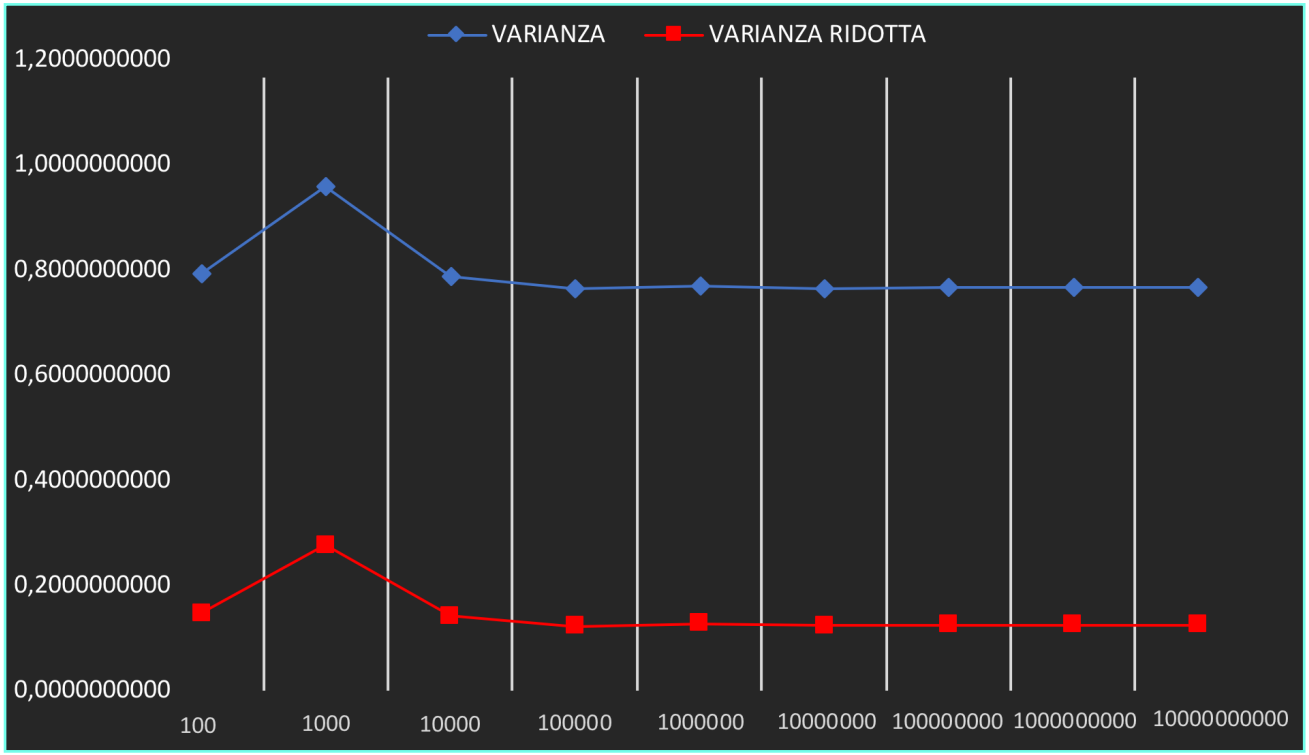
Per tanto ci concentremo a rappresentare i dati sperimentali di approfondimento circa:
INTERVALLO DI CONFIDENZA
VARIANZA,
VARIANZA RIDOTTA MEDIANTE VARIABILI ANTITETICHE,
DEVIAZIONE STANDARD PER ENTRAMBE LE VARIANZE STIMATE
esclusivamente per il Metodo Monte Carlo HIT-or-MISS.

INTERVALLO DI CONFIDENZA
METODO MONTE CARLO HIT-or-MISS

Dati sperimentali:

100	2,559458708	2,855141292
1000	2,5904432760	2,6832367240
10000	2,6955339820	2,7244860180
100000	2,7149264250	2,7240635750
1000000	2,7153502990	2,718541381
10000000	2,7183624740	2,7192763200
100000000	2,7182005050	2,7184895180
1000000000	2,718186859	2,7183782530
10000000000	2,718113837	2,7183427390
N_Eventi	**Intervallo inferiore**	**Intervallo superiore**

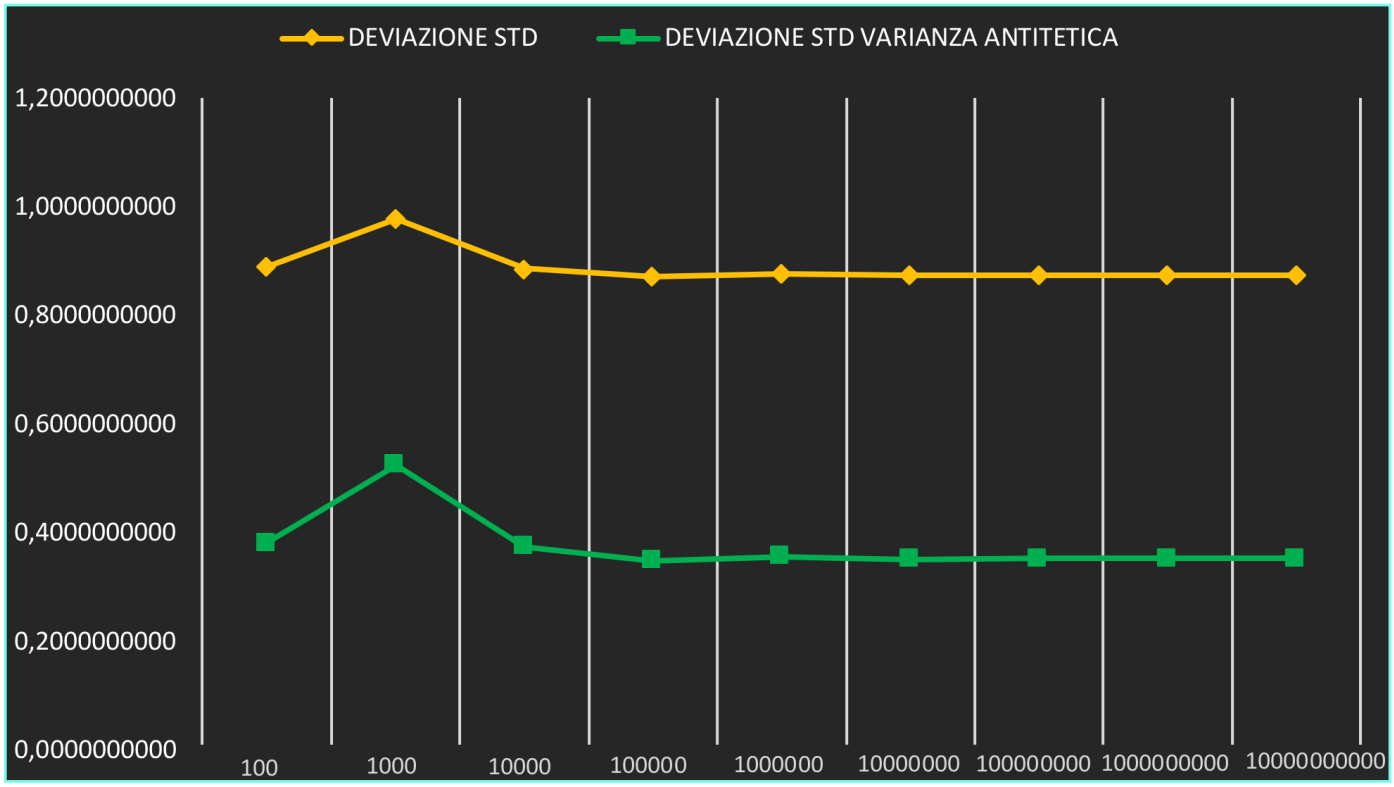
VARIANZA A CONFRONTO
METODO MONTE CARLO HIT-or-MISS



Dati sperimentali:

100	0,7924267100	0,1460767100
1000	0,9575948144	0,2760148144
10000	0,7858757999	0,1408807999
100000	0,7628319450	0,1225794450
1000000	0,7626078838	0,1224058038
10000000	0,7624792775	0,122288976
100000000	0,762235433	0,1220079387
1000000000	0,7620657822	0,1219320603
10000000000	0,7618551831	0,121830327
N_Eventi	**VARIANZA**	**VARIANZA_ANTITETICA**

DEVIAZIONE STANDARD A CONFRONTO
METODO MONTE CARLO HIT-or-MISS



Dati sperimentali:

100	0,8901835260	0,3821998299
1000	0,9785677362	0,5253711206
10000	0,8864963620	0,3753409116
100000	0,8734025103	0,3501134744
1000000	0,8771589843	0,3574993759
10000000	0,8743450563	0,3519786584
100000000	0,8750059617	0,3532816705
1000000000	0,8750233038	0,3533158082
10000000000	0,8750292470	0,3533275067
N_Eventi	**DEV_STD**	**DEV_STD_ANTITETICA**

CONCLUSIONE 2

Le tre pagine precedenti mostrano risultati circa:

- ***L'intervallo di confidenza****: Si può notare come inizialmente l'intervallo sia molto ampio, considerando tuttavia che essendo un numero di eventi di simulazione ridotto a 100, tecnicamente non è per nulla negativo il risultato. Ovviamente con l'aumentare del numero di eventi l'intervallo si va riducendo sempre di più in maniera sensibile, infatti con un numero di eventi pari a 10000000000 [10 miliardi] l'intervallo risulta essere cosi piccolo da variare dalla quarta cifra dopo la virgola quindi un risultato veramente ottimo!*

- ***Varianza e Varianza Ridotta****: Il grafico a pagina 20 è molto chiaro, la varianza ridotta mediante l'utilizzo delle variabili antitetiche è molto più piccola rispetto alla varianza normale. Comunque sia possiamo notare che in entrambi i casi essa rimane piuttosto costante anche se, ovviamente, all'aumentare del numero di eventi essa si va riducento di molto poco per entrambi.*

- ***Deviazione Standard e Deviazione Standard a Varianza Ridotta****: Anche in questo caso il grafico al pagina 21 è molto esplicativo: la deviazione standard nel caso della varianza che è stata ridotta è molto minore rispetto alla deviazione standard con varianza normale. Possiamo inoltre dire che per entrambi i casi l'andamento di linerarità è molto simile.*

CONCLUSIONE FINALE

Considerando tutte le prove e i test sperimentali possiamo concludere che complessivamente l'algoritmo HIT-or-MISS appartenente alla famiglia dei Metodi Monte Carlo risulta un ottimo algoritmo per determinare il valore di Nepero messo anche in confronto ad un altro algoritmo alternativo stocastico i cui risultati e tempi di esecuzione sono nettamente inferiori rispetto al primo.