MonteCarlo_AHI

April 3, 2023

0.1 Exercício 1 - Método de Monte Carlo

1. Alexis Duarte Guimarães Mariz | 2019006337
2. Henrique Rotsen Santos Ferreira | 2020100945
3. Ivan Vilaça de Assis | 2021421931

[ ]:
```python
# Imports

import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
```

0.2 Definições das Funções

Aqui vamos definir as nossas funções para realizarem os 2 métodos de Monte Carlo, chamaremos de `MonteCarlo1(a, b, ymax, n, N, fun)`, em que: * **a**: É o valor *inicial* da integral, * **b**: É o valor *final* da integral * **ymax**: É o valor máximo que a função atinge, * **n**: É o Nº total de números aleatórios gerados, * **N**: É o Nº de amostras geradas, * **fun**: É a função que está sendo testada.

[ ]:
```python
def MonteCarlo1(a, b, ymax, n, N, fun):
    lado1 = b-a
    lado2 = ymax
    lista = []
    for i in range(N):
        cont = 0
        for j in range(n):
            x = np.random.rand(1)[0]*b
            y = np.random.rand(1)[0]*ymax

            if y < fun(x):
                cont +=1
        lista.append(lado1*lado2*cont/n)

    err = np.std(lista)/np.sqrt(N)
    plt.hist(lista)
    resp = sum(lista)/N
    plt.axvline(resp, color='r', linewidth=1)
    plt.title("N Total: " + str(n))
    plt.xlabel("Valor da Integral")
```

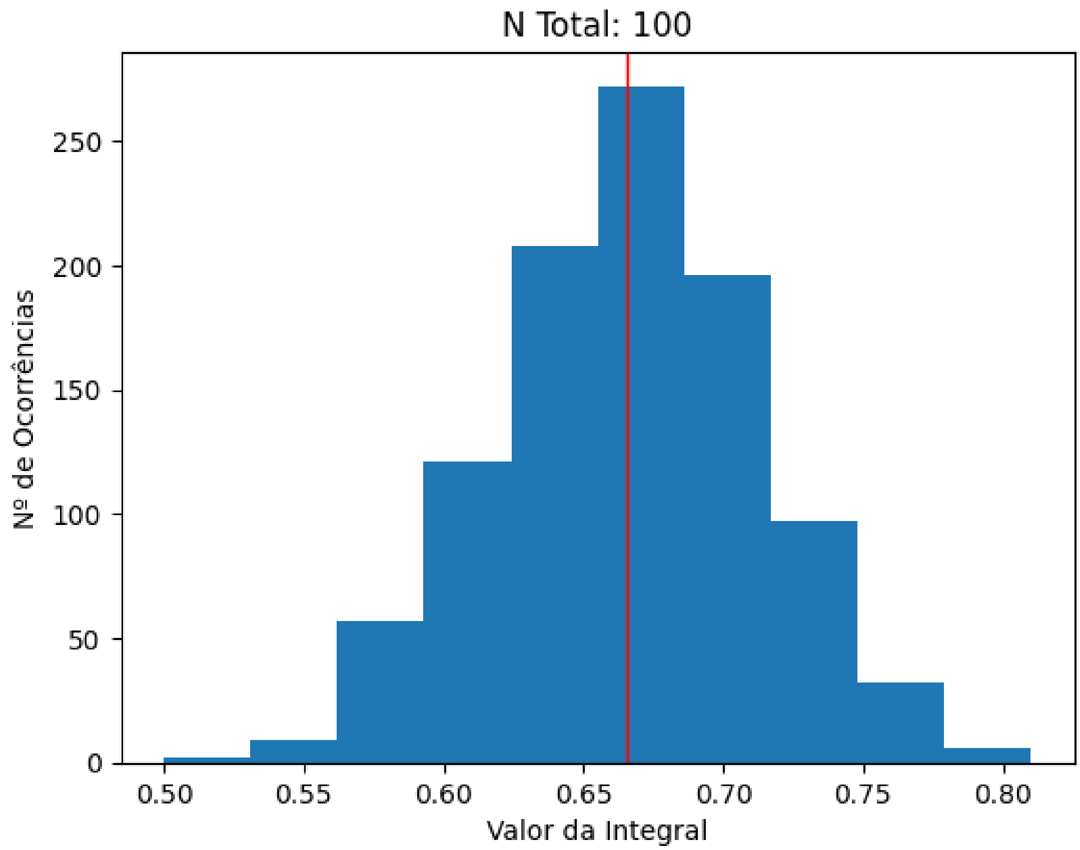
```
    plt.ylabel("Nº de Ocorrências")
    plt.show()
    print("Valor Médio da integral = " + str(resp))
    print("Erro padrão = " + str(err))
```

Já o segundo método, chamaremos de `MonteCarlo2(a, b, n, N, fun)`, que segue todo o pensamento do primeiro, porém não utilizamos *ymax*

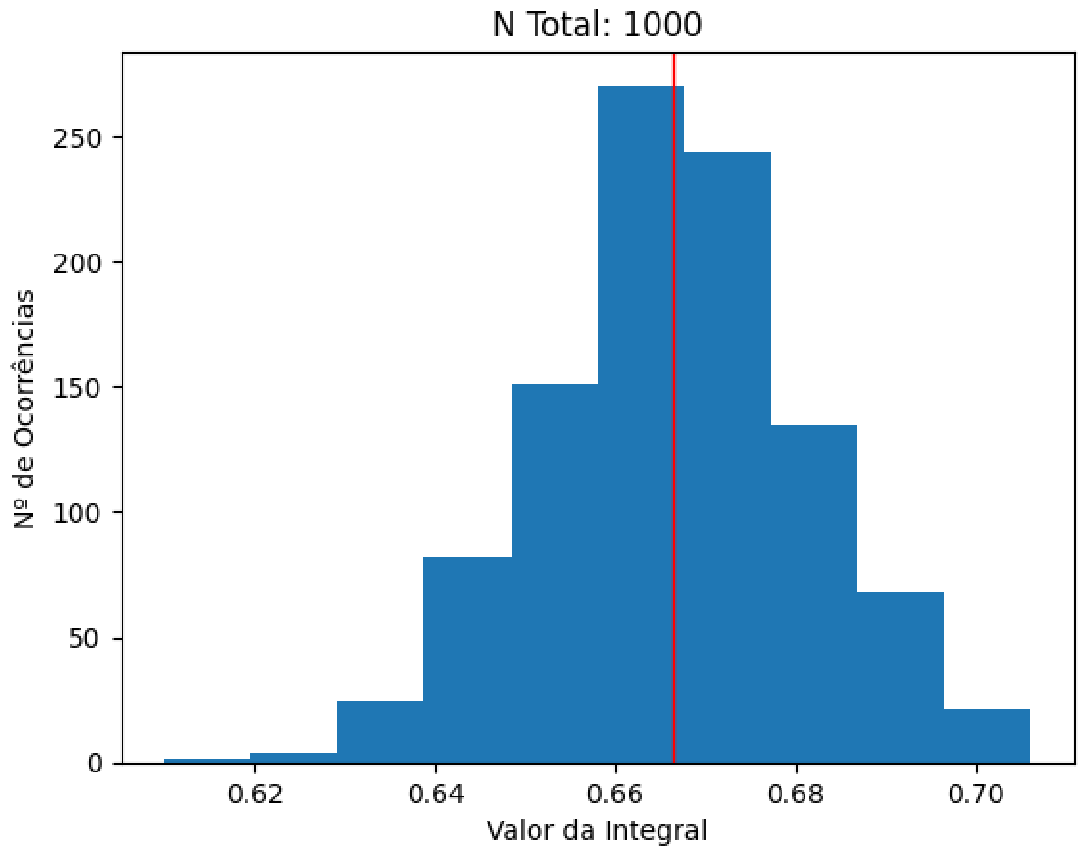
[ ]:
```
def MonteCarlo2(a, b, n, N, fun):
    lista = []
    for i in range(N):
        somatorio = 0
        for j in range(n):
            x = np.random.rand(1)[0]*b
            somatorio += fun(x)
        lista.append(((b-a)/n)*somatorio)

    err = np.std(lista)/np.sqrt(N)
    plt.hist(lista)
    resp = sum(lista)/N
    plt.axvline(resp, color='r', linewidth=1)
    plt.title("N Total: " + str(n))
    plt.xlabel("Valor da Integral")
    plt.ylabel("Nº de Ocorrências")
    plt.show()
    print("Valor Médio da integral = " + str(resp))
    print("Erro padrão = " + str(err))
```

0.3 Exercício 1

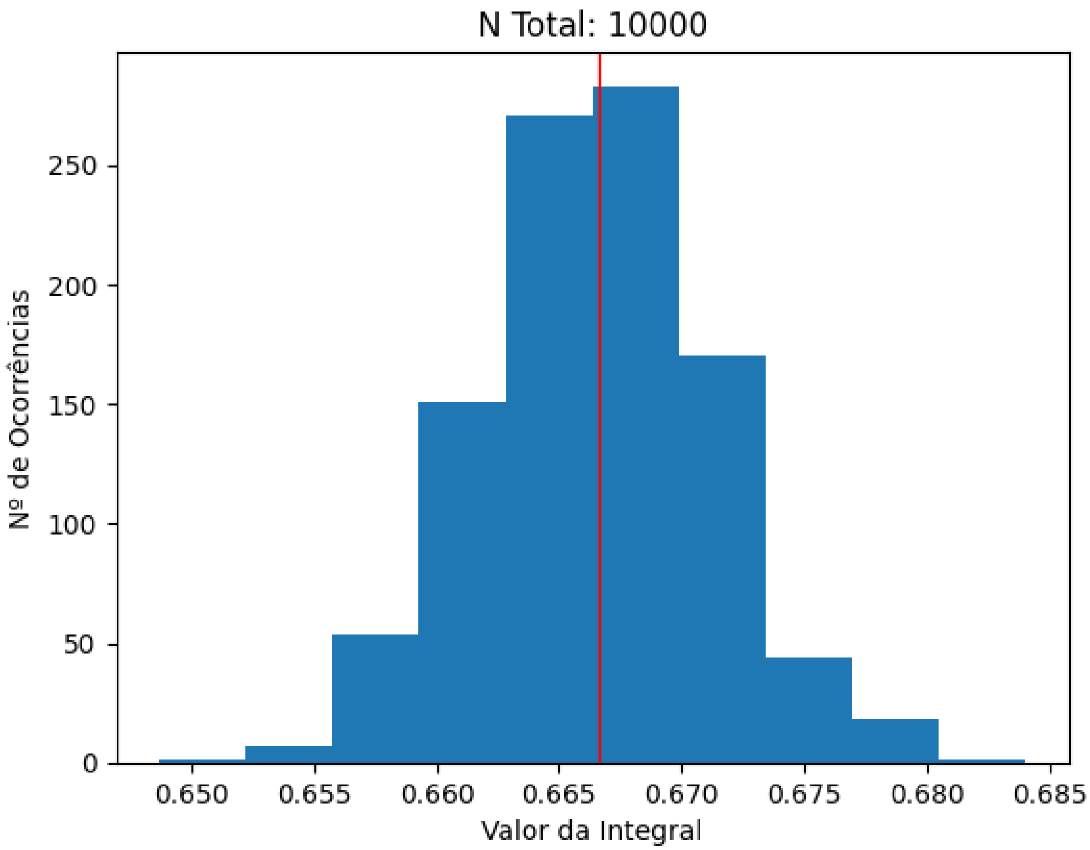
[ ]:
```
#Metodo 1

def fun1(x):
    return 1-x*x

MonteCarlo1(0,1,1,100,1000, fun1)
MonteCarlo1(0,1,1,1000,1000, fun1)
MonteCarlo1(0,1,1,10000,1000, fun1)
```




```
Valor Médio da integral = 0.6653999999999998
Erro padrão = 0.0014385548303766525
```



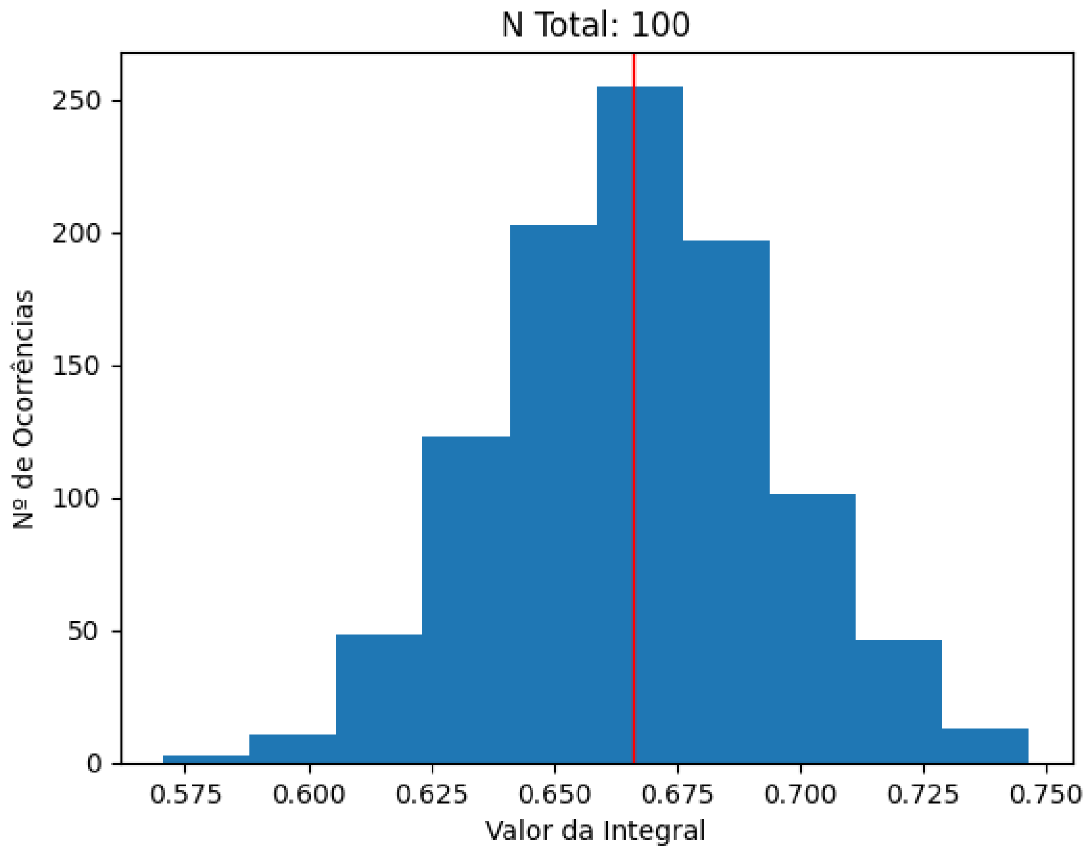

```
Valor Médio da integral = 0.6664179999999996
Erro padrão = 0.0004629376588699605
```



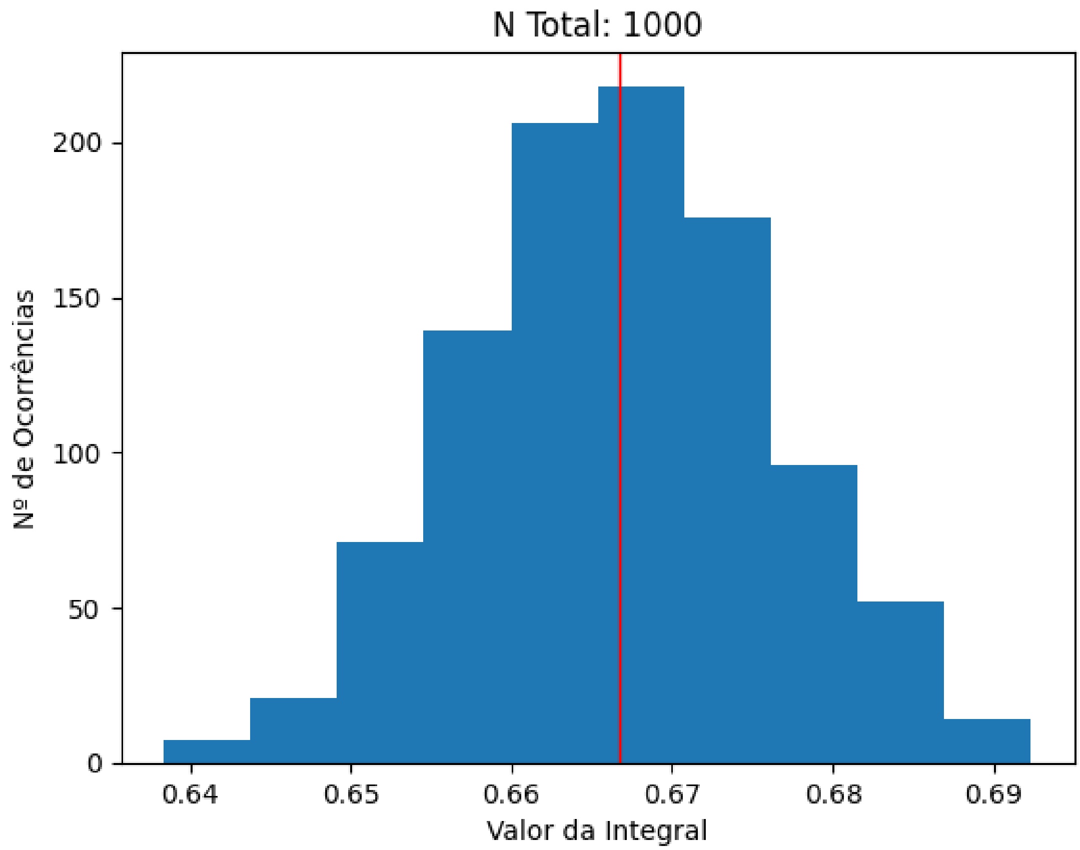

```
Valor Médio da integral = 0.6665932999999998
Erro padrão = 0.0001477442557597418
```

[ ]:

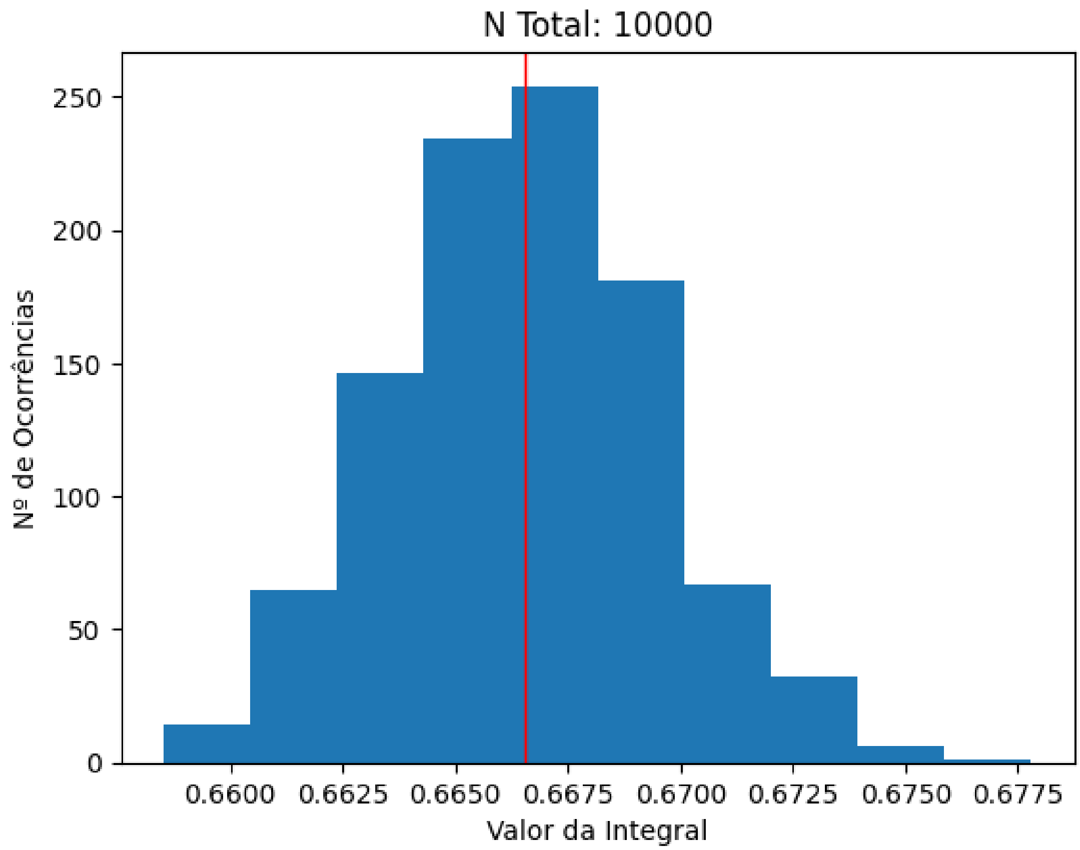
```
#Metodo 2

MonteCarlo2(0,1,100,1000, fun1)
MonteCarlo2(0,1,1000,1000, fun1)
MonteCarlo2(0,1,10000,1000, fun1)
```




```
Valor Médio da integral = 0.6661645110796475
Erro padrão = 0.0008966407159152026
```




```
Valor Médio da integral = 0.6667509029322567
Erro padrão = 0.00029541557432161066
```



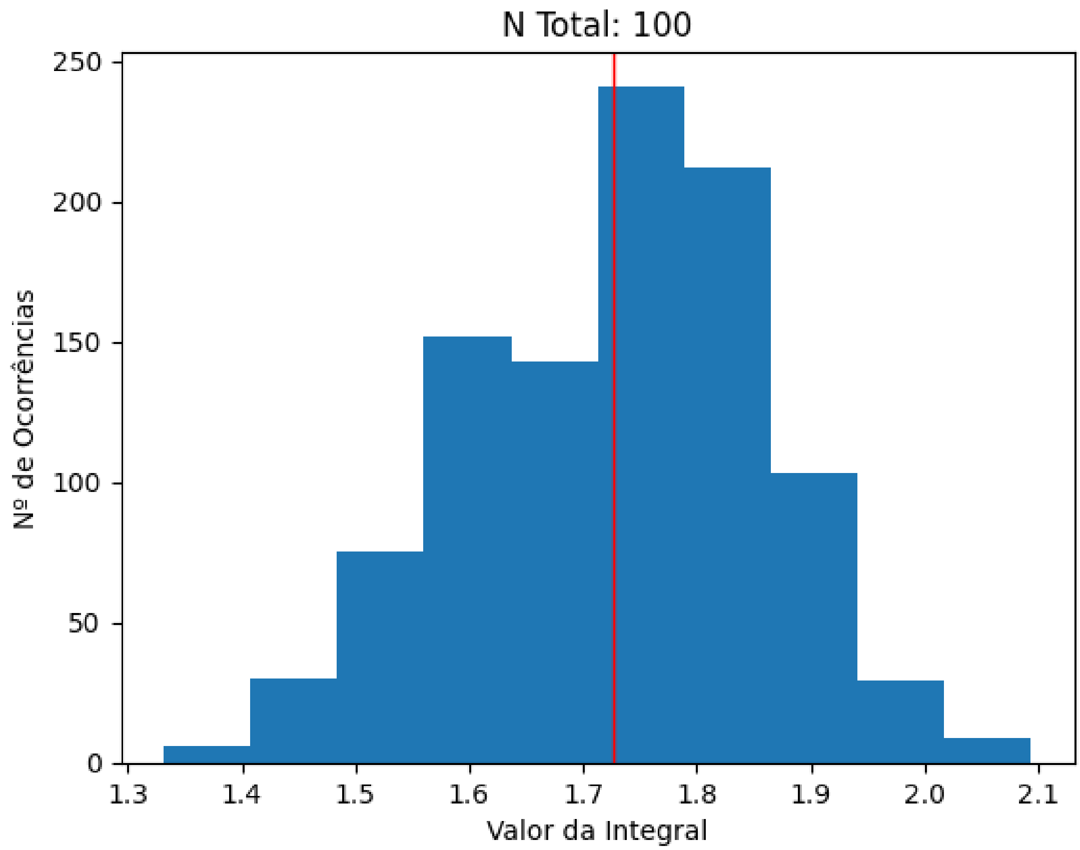

```
Valor Médio da integral = 0.6665476751669355
Erro padrão = 9.255259861460324e-05
```

Após rodarmos o Método 1, pode-se ver que o resultado foi, desde a primeira amostra, como o esperado. Dando continuidade, quanto mais aumentamos o número de amostras, menor o **erro** e maior a precisão em casas decimais do Valor Médio da Integral.

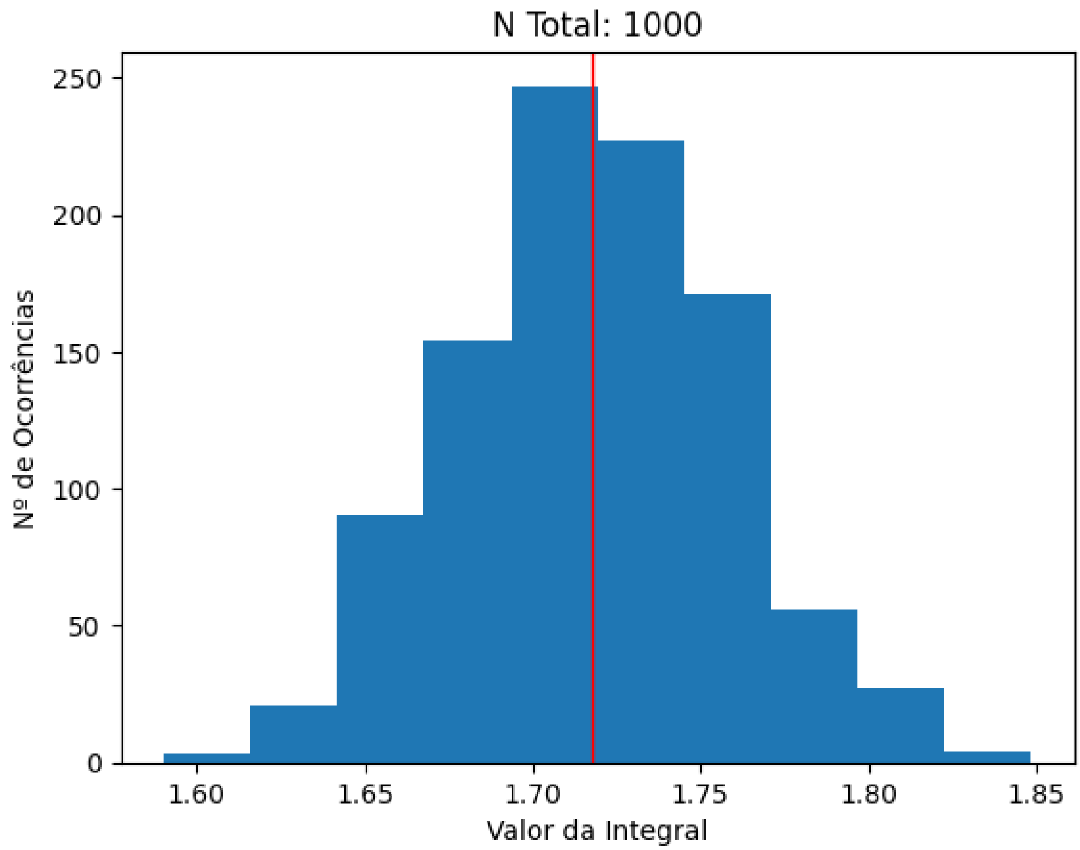
Já no Método 2 atingimos um resultado ainda melhor que antes, e de maneira mais eficiente, pois o tempo de execução caiu quase pela metade.

0.4 Exercício 2

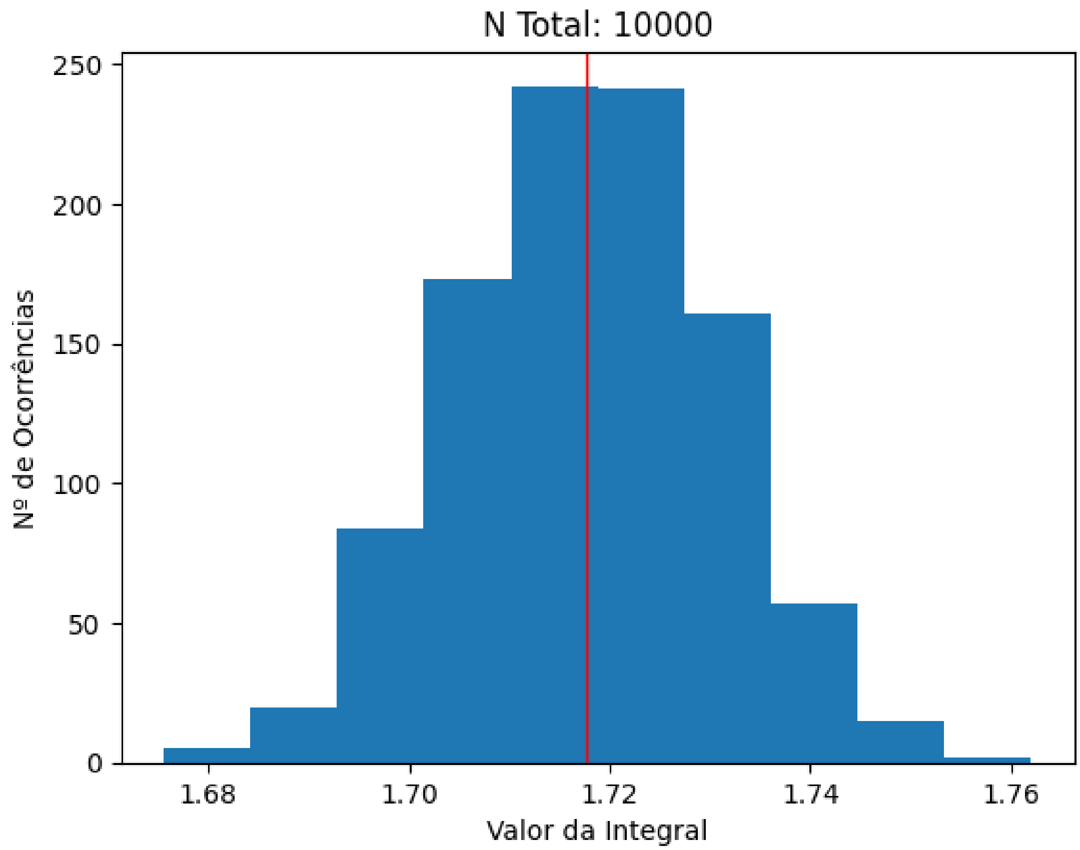
[ ]:
```
#Metodo 1

def fun2(x):
    return np.exp(x)

MonteCarlo1(0,1,np.e,100,1000, fun2)
MonteCarlo1(0,1,np.e,1000,1000, fun2)
MonteCarlo1(0,1,np.e,10000,1000, fun2)
```




```
Valor Médio da integral = 1.725973046980073
Erro padrão = 0.0040754265821550595
```

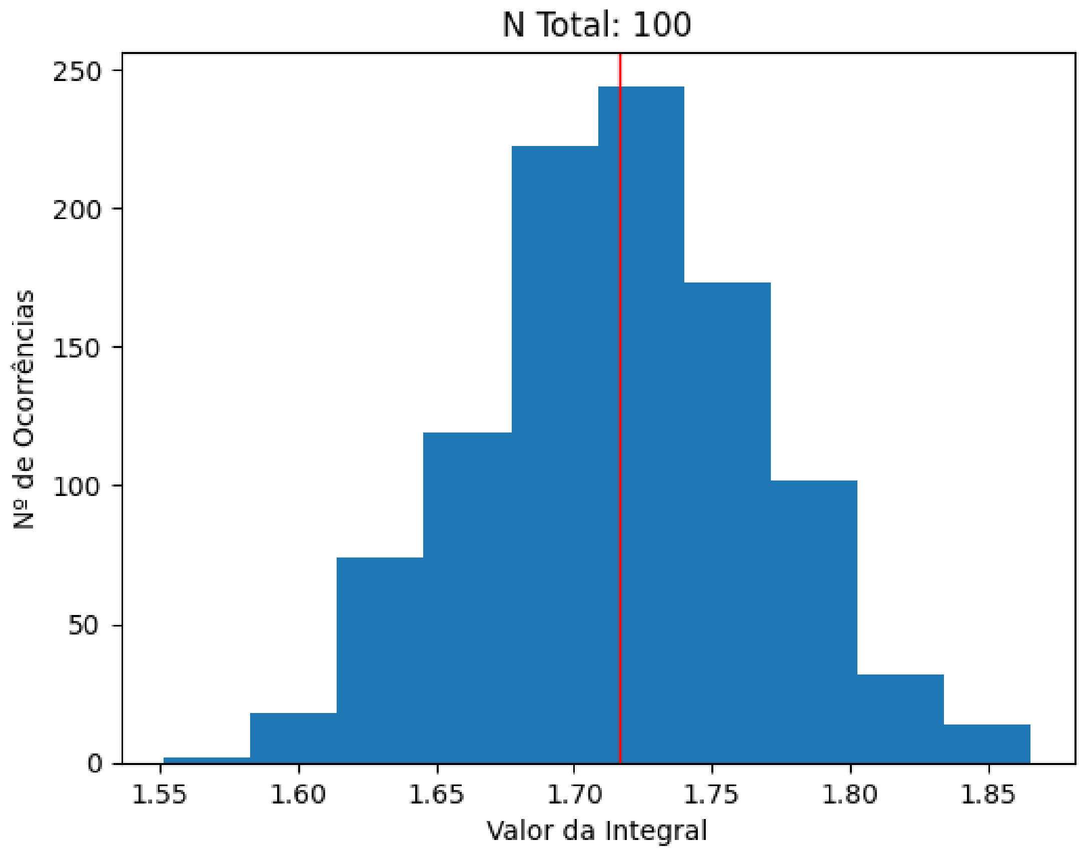



```
Valor Médio da integral = 1.7176877239669253
Erro padrão = 0.0012747507070917225
```

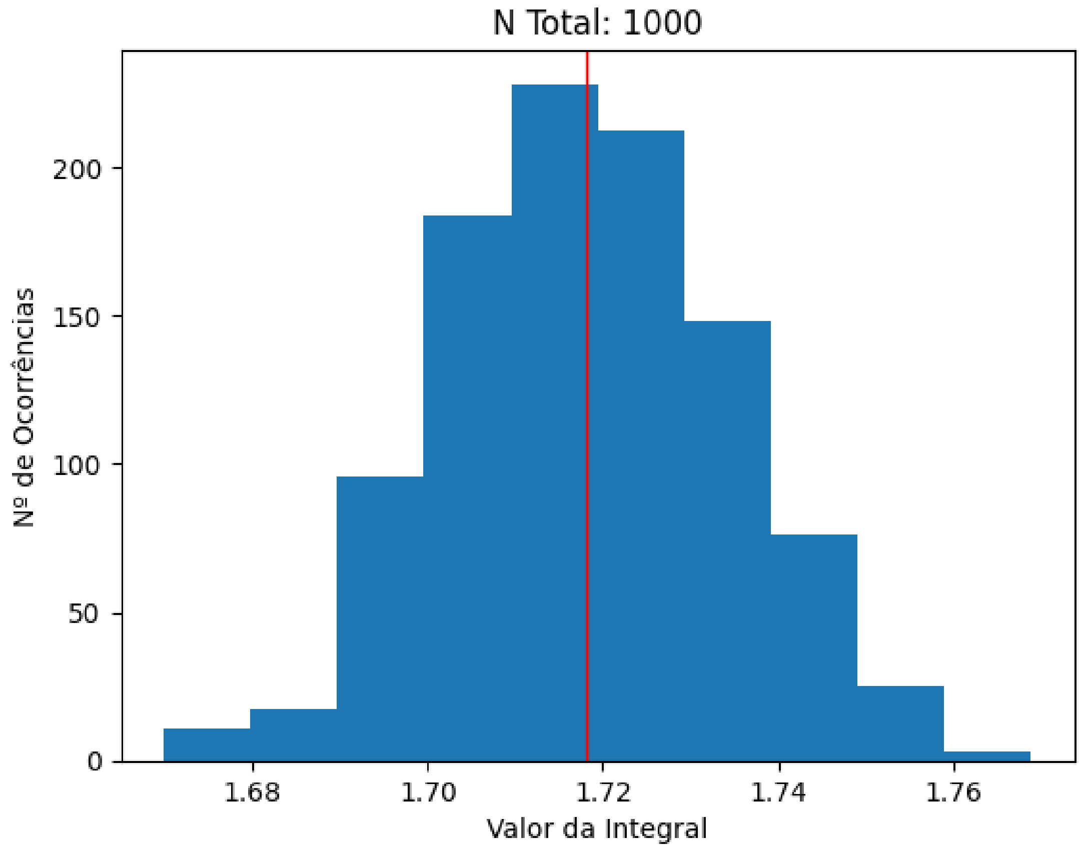



```
Valor Médio da integral = 1.7177143631288483
Erro padrão = 0.0004094833384693844
```

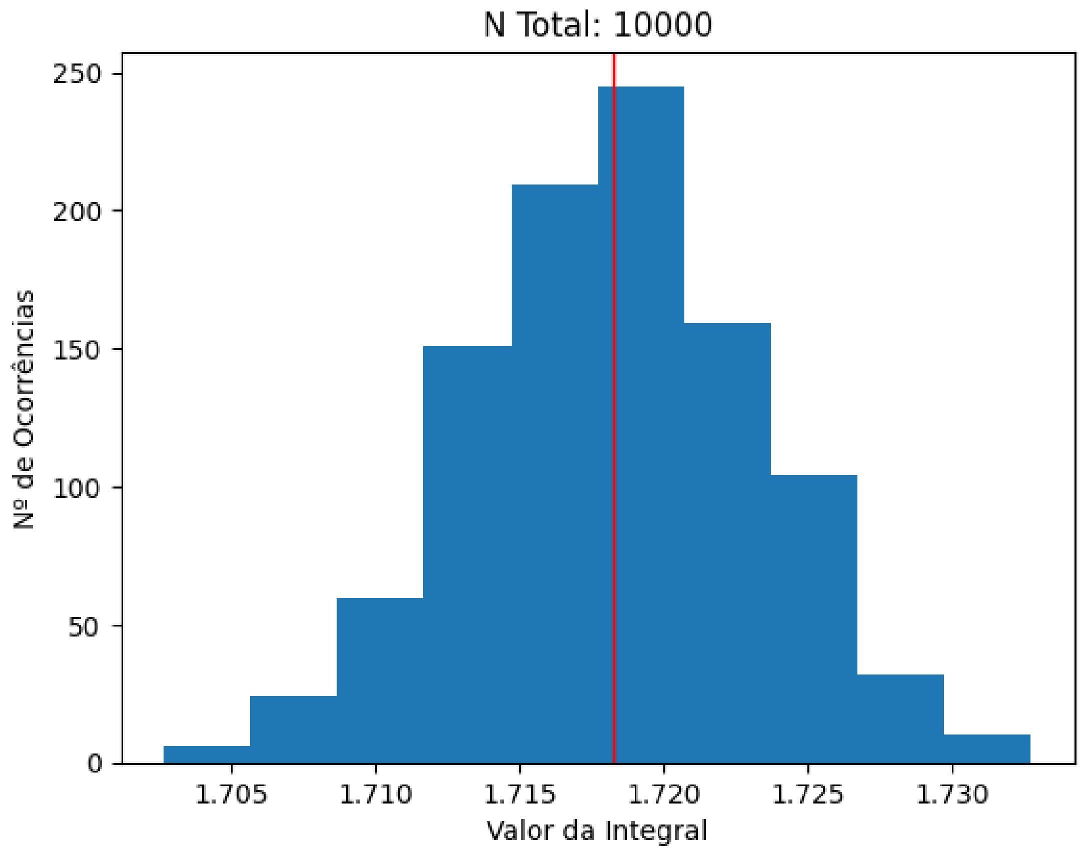
[ ]:
```
#Metodo 2

MonteCarlo2(0,1,100,1000, fun2)
MonteCarlo2(0,1,1000,1000, fun2)
MonteCarlo2(0,1,10000,1000, fun2)
```




```
Valor Médio da integral = 1.716560436180095
Erro padrão = 0.00161439711304177
```




```
Valor Médio da integral = 1.7182310541101817
Erro padrão = 0.0005072492735599297
```

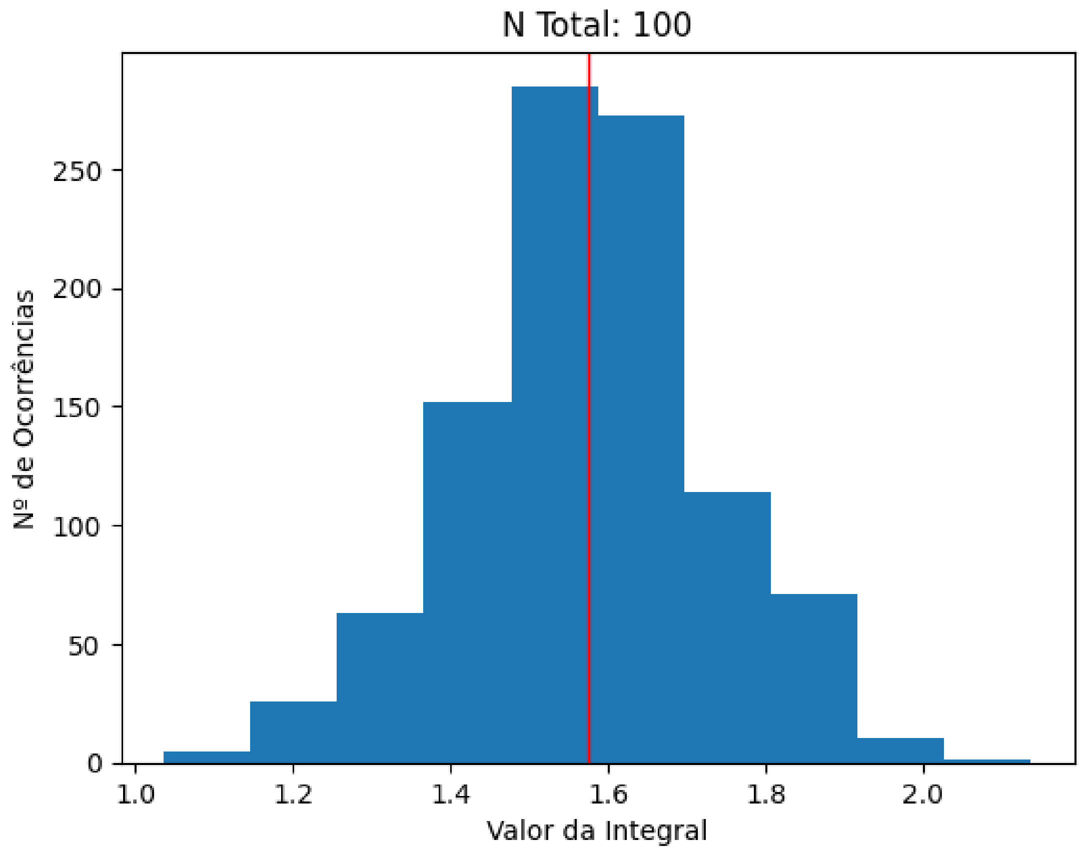



```
Valor Médio da integral = 1.7182395675050883
Erro padrão = 0.00015696872005715591
```

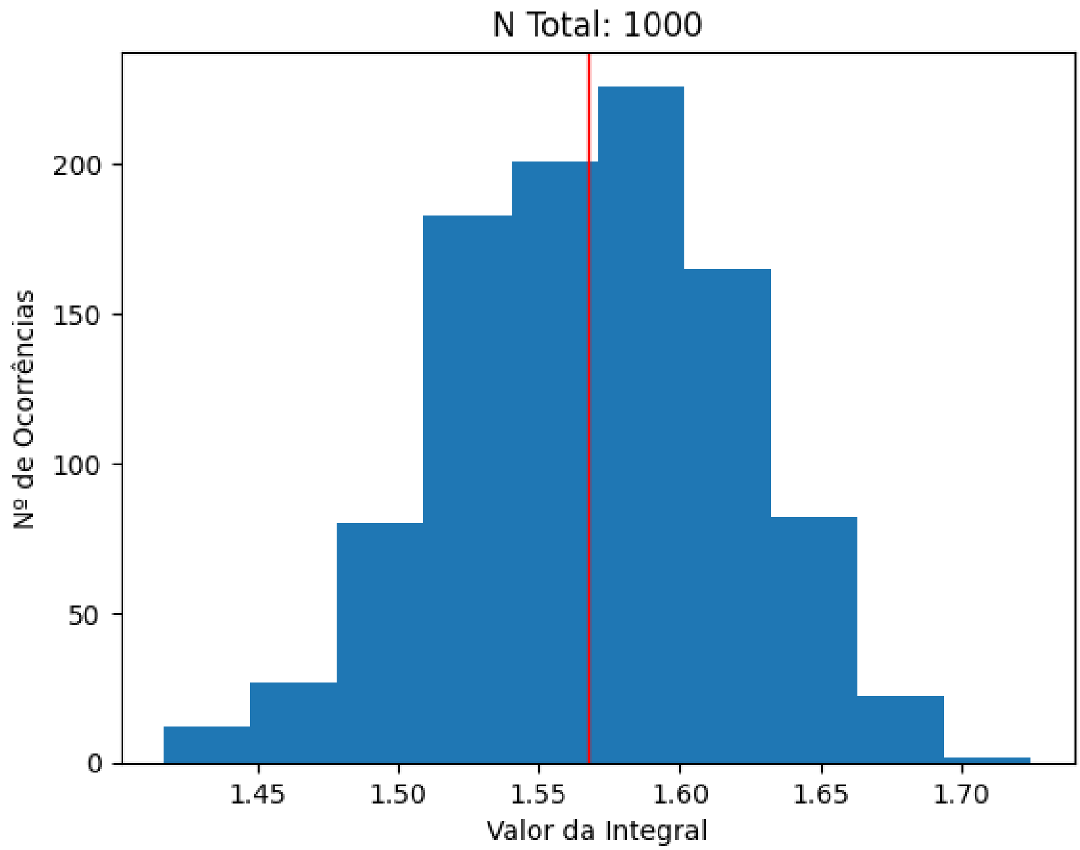
Para este exercício pode-se ter basicamente a mesma análise que houve para o exercício anterior. Porém neste caso o **erro** não foi tão baixo quanto no Exercício 1, e ele se manteve da mesma ordem de grandeza nos 2 métodos, o que não aconteceu naquele.

0.5 Exercício 3

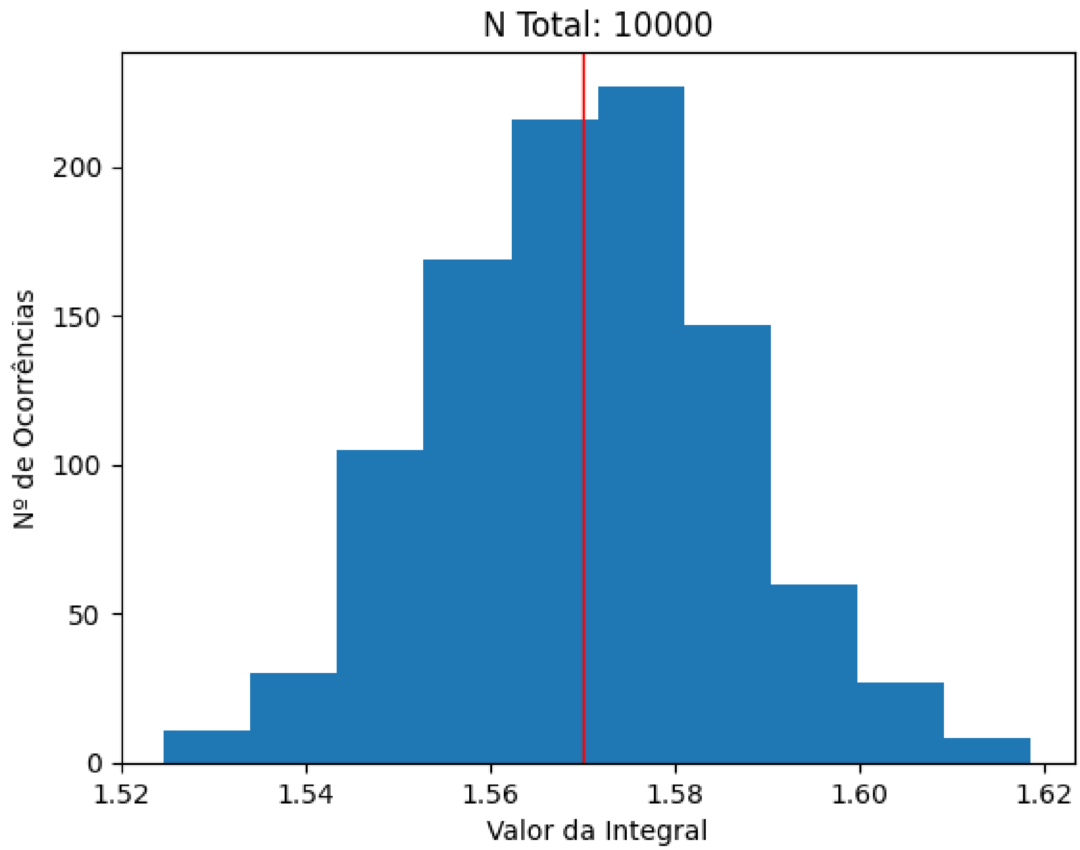
[ ]:
```
#Metodo 1

def fun3(x):
    return np.sin(x)**2

MonteCarlo1(0,np.pi,1,100,1000, fun3)
MonteCarlo1(0,np.pi,1,1000,1000, fun3)
MonteCarlo1(0,np.pi,1,10000,1000, fun3)
```




```
Valor Médio da integral = 1.574880397244563
Erro padrão = 0.005158634804161006
```

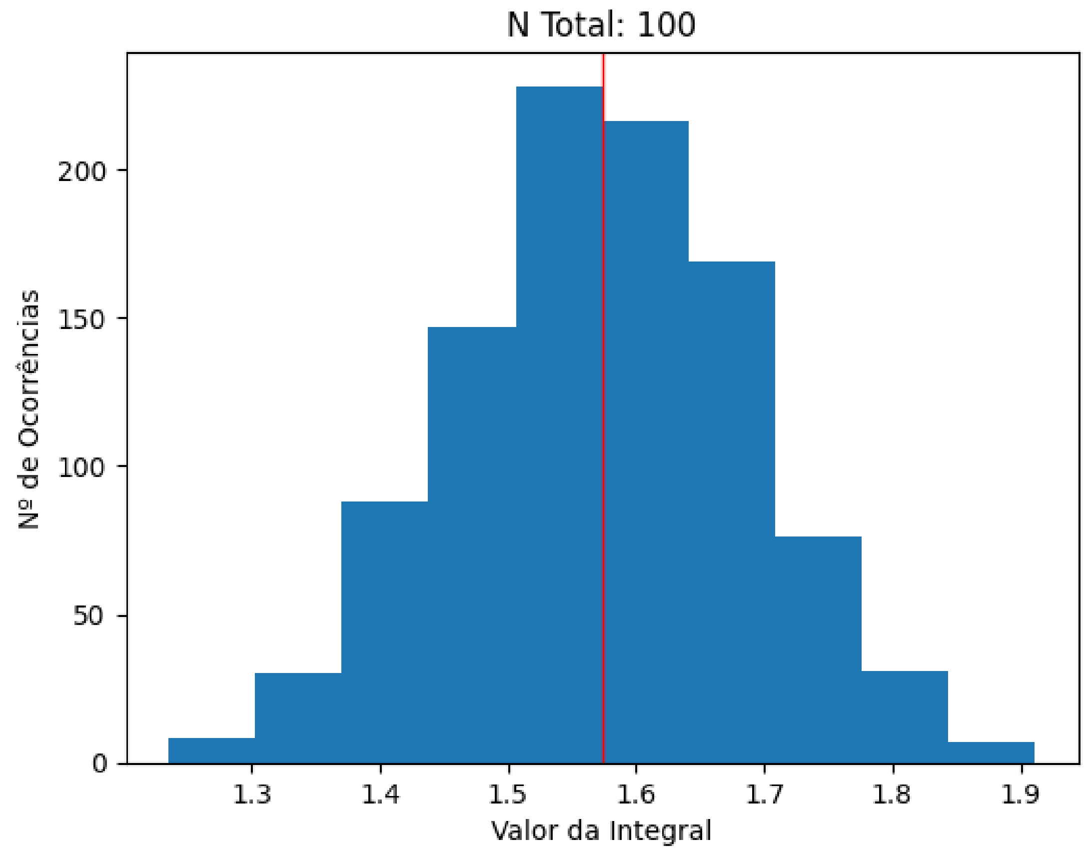



```
Valor Médio da integral = 1.567940619072785
Erro padrão = 0.0015930096609417222
```

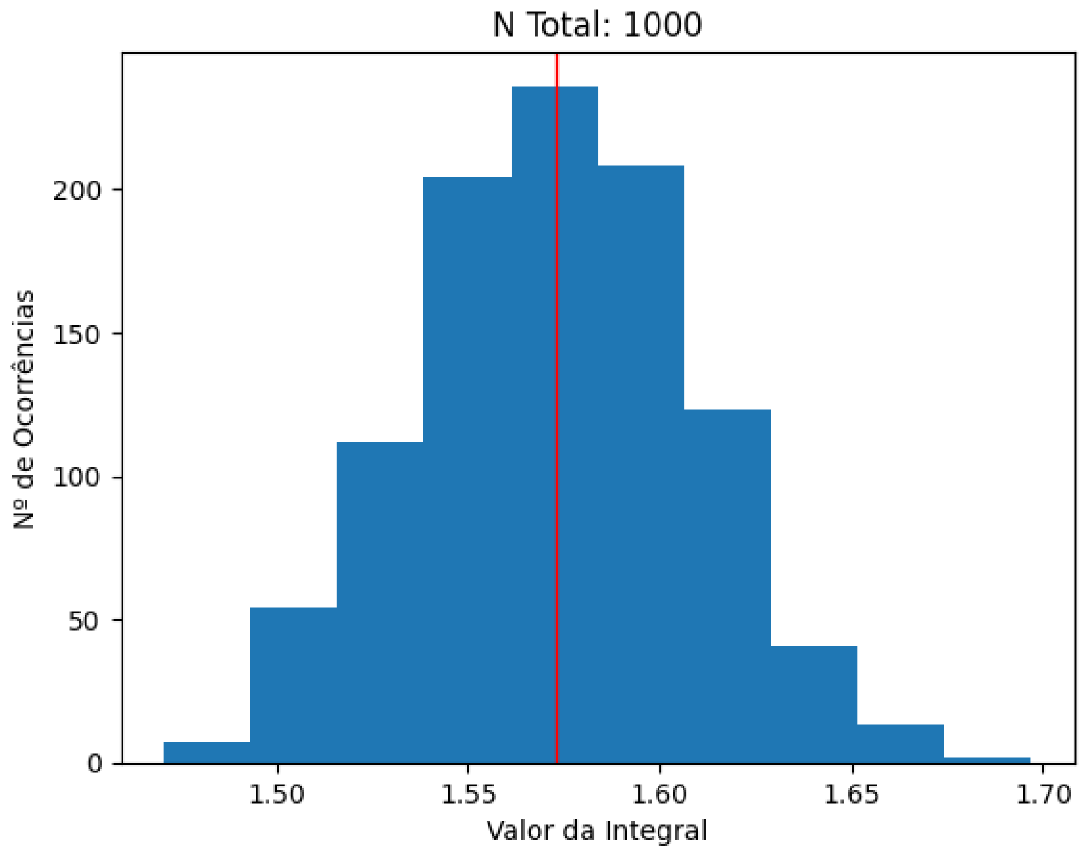



```
Valor Médio da integral = 1.5700797295106137
Erro padrão = 0.0005038811062576503
```

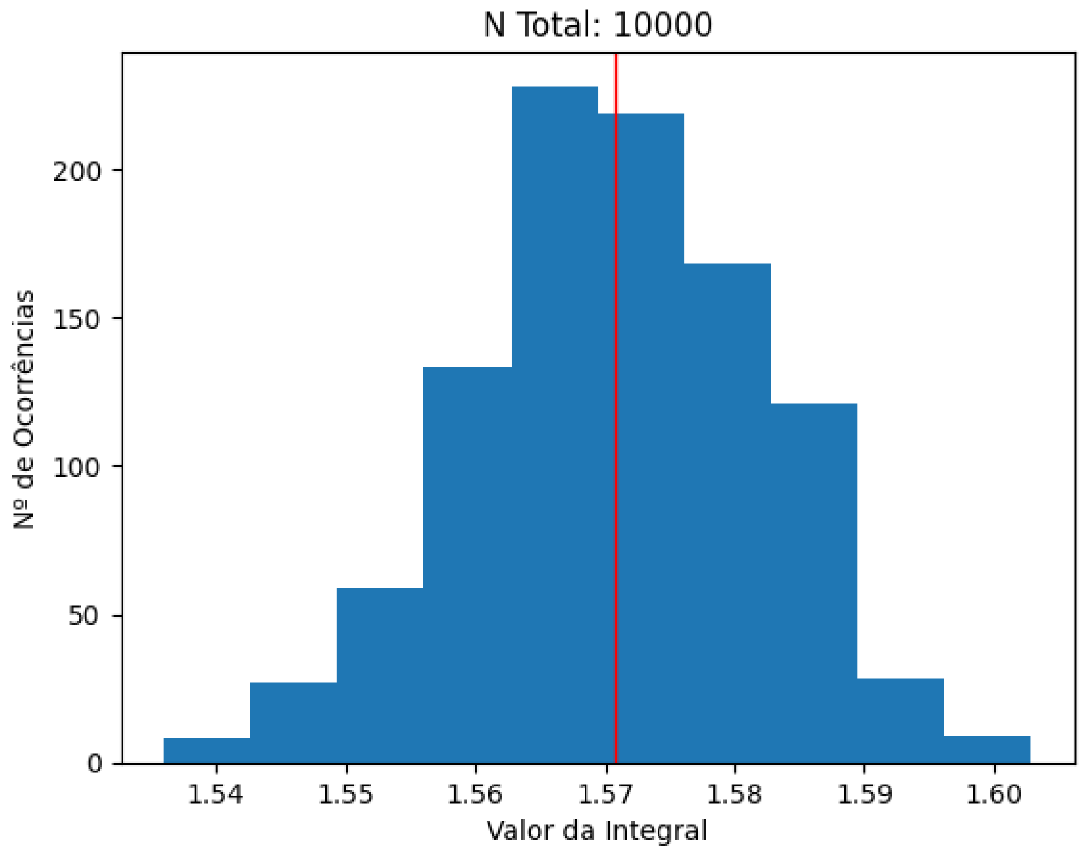
[ ]:
```
#Metodo 2

MonteCarlo2(0,np.pi,100,1000, fun3)
MonteCarlo2(0,np.pi,1000,1000, fun3)
MonteCarlo2(0,np.pi,10000,1000, fun3)
```




```
Valor Médio da integral = 1.5731182495168559
Erro padrão = 0.0036263727020417855
```




```
Valor Médio da integral = 1.5729038460925153
Erro padrão = 0.001130299481826559
```




```
Valor Médio da integral = 1.5708469342568525
Erro padrão = 0.00035407698436600254
```

Para este exercício, pode-se ter exatamente a mesma análise do anterior, uma vez que os resultados obtidos bateram com a resposta esperada e não houve grande divergência entre os métodos.

0.6 Exercício 4

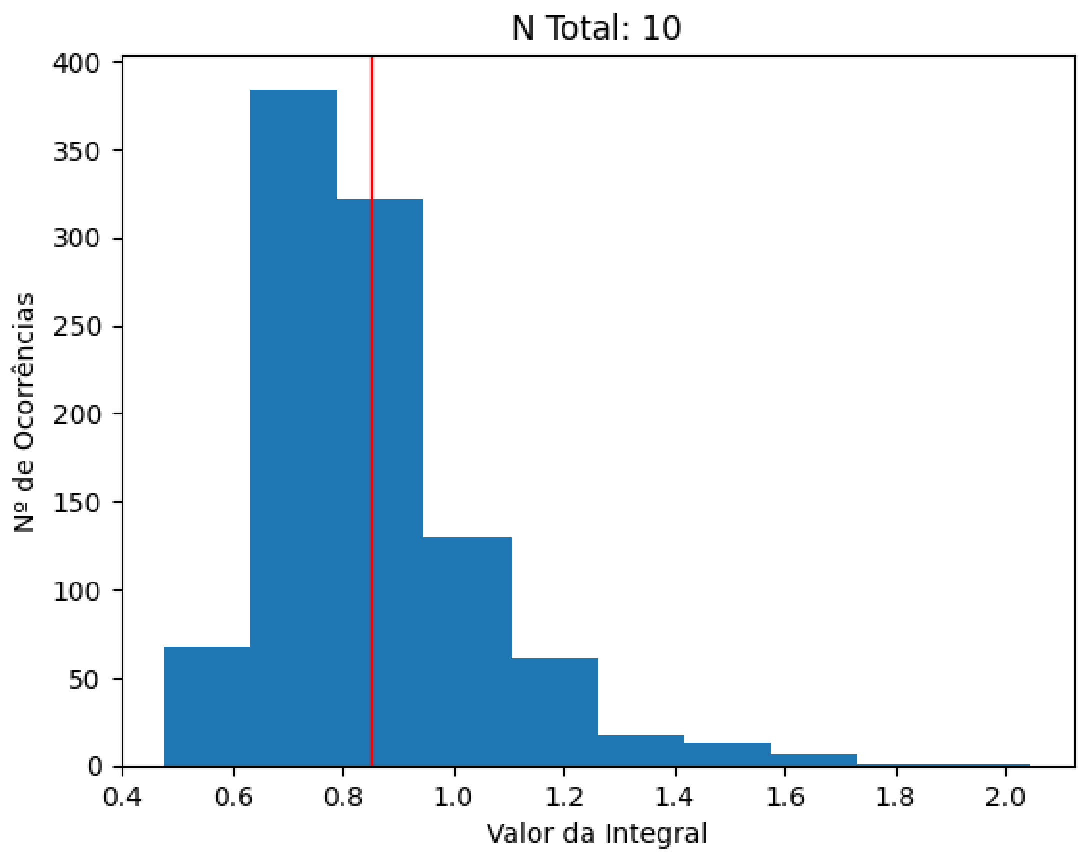
[ ]:
```
def MonteCarlo2_XD(a, b, n, N, fun, D):
    lista = []
    for i in range(N):
        somatorio = 0
        for j in range(n):
            rList = np.random.rand(D)*b
            somatorio += fun(rList)
        lista.append(((b-a)/n)*somatorio)

    err = np.std(lista)/N
    print(err)
    plt.hist(lista)
    resp = sum(lista)/N
```

```
    plt.axvline(resp, color='r', linewidth=1)
    plt.title("N Total: " + str(n))
    plt.xlabel("Valor da Integral")
    plt.ylabel("Nº de Ocorrências")
    plt.show()
    print("Valor Médio da integral = " + str(resp))
```

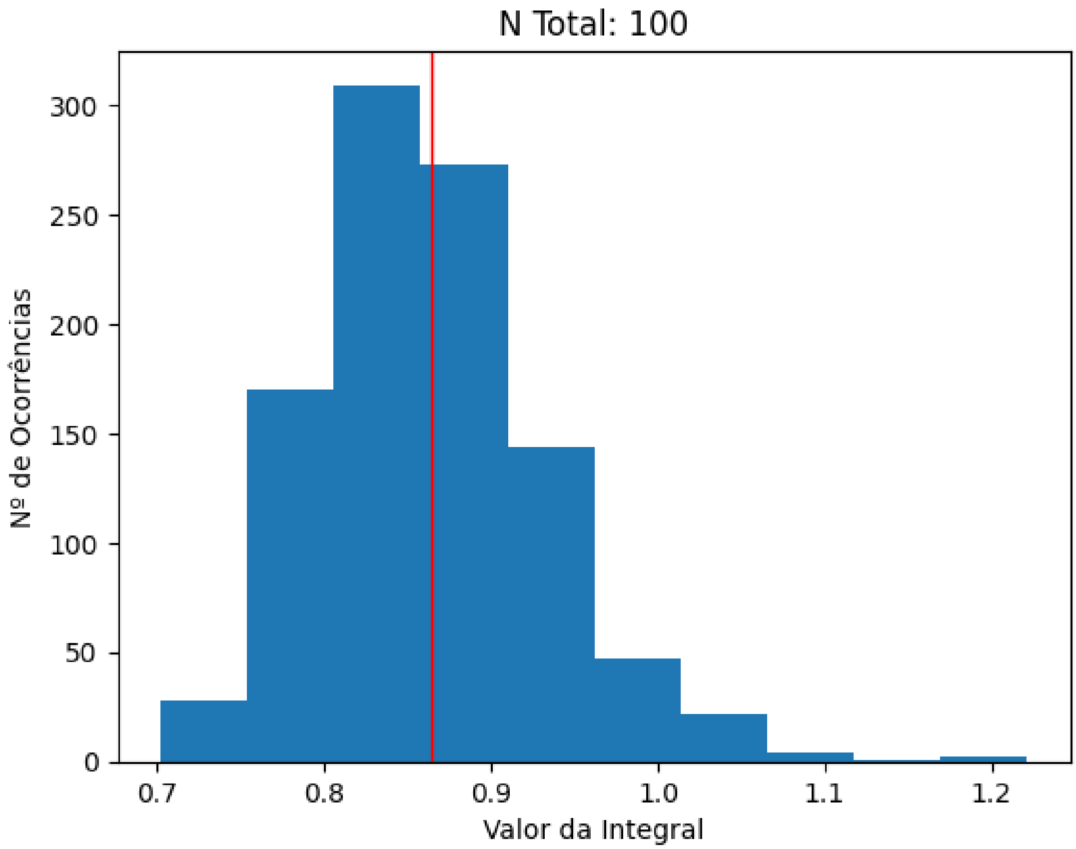
[ ]:
```
# Metodo 2
def fun4(l):
    x = (l[0]+l[1])*l[2]
    y = (l[3]+l[4])*l[5]
    z = (l[6]+l[7])*l[8]
    return 1/(x+y+z)
```

[ ]:
```
MonteCarlo2_XD(0,1,10,1000,fun4,9)
MonteCarlo2_XD(0,1,100,1000,fun4,9)
MonteCarlo2_XD(0,1,1000,1000,fun4,9)
MonteCarlo2_XD(0,1,10000,1000,fun4,9)
```

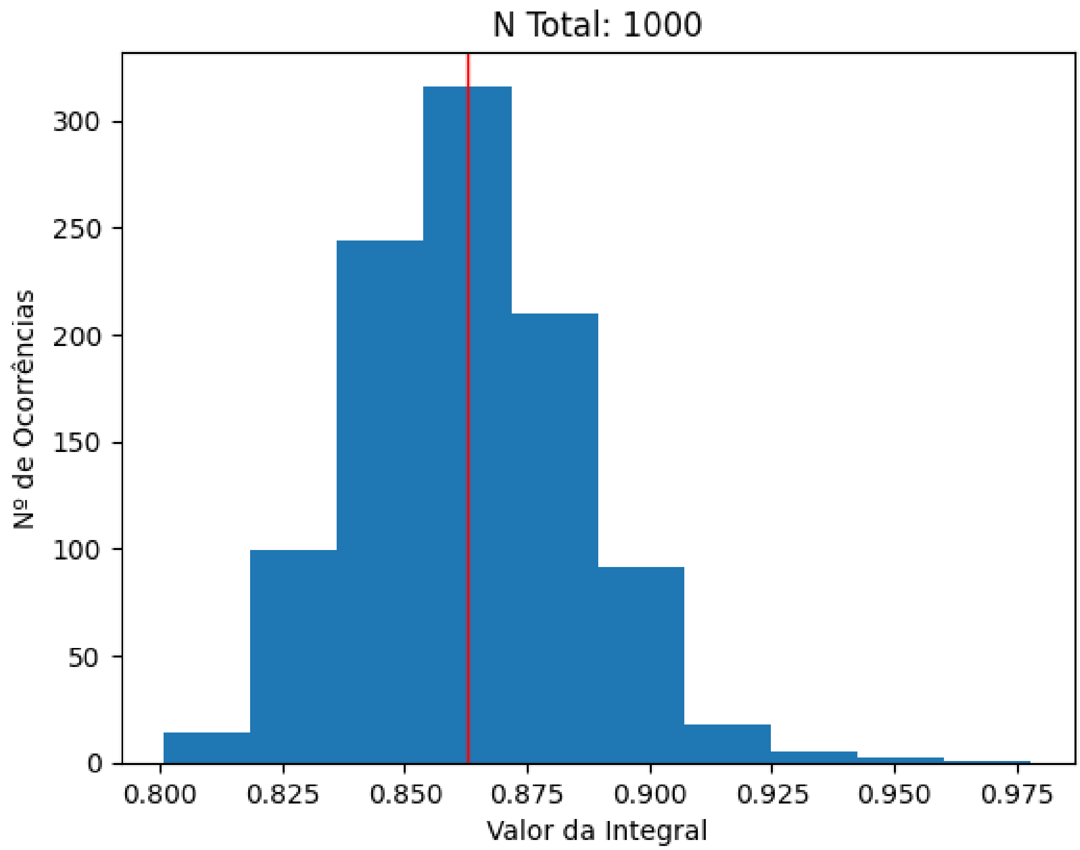
```
0.0001953007879572533
```



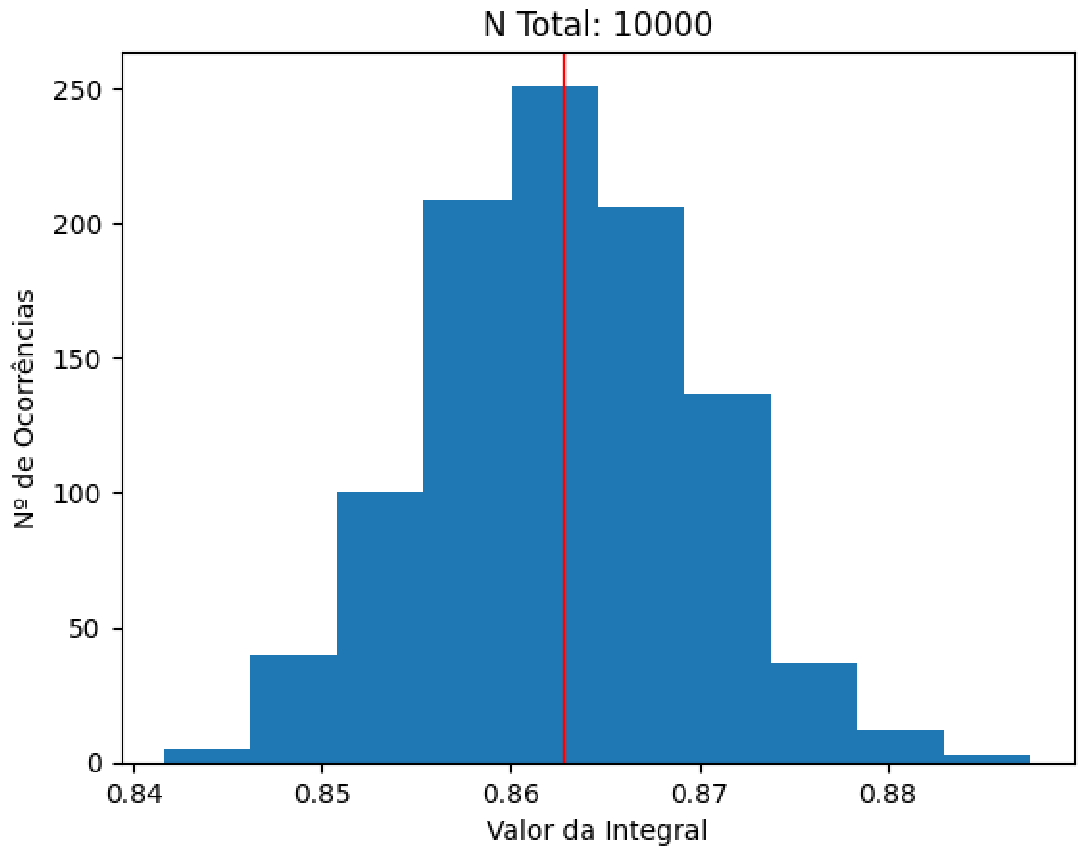
```
Valor Médio da integral = 0.8498606609725814
6.858553053759e-05
```




```
Valor Médio da integral = 0.8640344010293367
2.2267809227494762e-05
```




```
Valor Médio da integral = 0.8628539520429136
6.926187004579782e-06
```



`Valor Médio da integral = 0.8627801309066321`

Neste exercício temos algo um pouco diferente, para valores pequenos de NTotal, temos uma divergência que pode ser observada no histograma. Os valores obtidos parecem *puxar* para a direita, talvez seja a geração dos números aleatórios. Porém para valores maiores de NTotal temos um histograma mais parecido com uma normal, centrada em 0.86, que parece ser o resultado correto.