exercicio-01-giuliavieira

December 12, 2023

UNIVERSIDADE FEDERAL DE MINAS GERAIS INSTUTUTO DE CIÊNCIAS EXATAS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DISCIPLINA: Introdução a Física Estatística e Computacional

ALUNA: Giulia Monteiro Silva Gomes Vieira MATRICULA: 2016006492

EXERCÍCIO AVALIATIVO 01: INTEGRAÇÃO DIRETA POR MONTE CARLO

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import random

from typing import Callable
```

[2]:
```
N_SAMPLES = 1000
```

Primeiro método

[3]:
```
def first_method(f, lim_inf, lim_sup, n_points):
    points_inside = 0

    for _ in range(n_points):
        x = random.uniform(lim_inf, lim_sup)
        y = random.uniform(0, max(f(lim_inf), f(lim_sup)))

        if 0 <= y <= f(x):
            points_inside += 1

    rectangular_area = (lim_sup - lim_inf) * max(f(lim_inf), f(lim_sup))
    fractional_points_inside = points_inside / n_points

    estimate_integral = rectangular_area * fractional_points_inside
    return estimate_integral
```

Segundo método

[4]:
```
def second_method(f, lim_inf, lim_sup, n_points):
    accumulator = 0

    for _ in range(n_points):
```

```
        x = random.uniform(lim_inf, lim_sup)
        accumulator += f(x)

    average = accumulator / n_points
    estimate_integral = (lim_sup - lim_inf) * average

    return estimate_integral
```

[5]:
```
def get_histogram(integral_func, method, lim_inf, lim_sup, n_points, n_samples):
    estimates = []

    for _ in range(n_samples):
        e = method(integral_func, lim_inf, lim_sup, n_points)
        estimates.append(e)

    return estimates
```

[6]:
```
def plot_histogram(estimates_method_1, estimates_method_2, n_points):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.hist(estimates_method_1, bins=30, color='blue', alpha=0.7,␣
 ↪label='Método 1')
    plt.title(f'Histograma - Método 1 (N = {n_points})')
    plt.xlabel('Estimativa da Integral')
    plt.ylabel('Frequência')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.hist(estimates_method_2, bins=30, color='green', alpha=0.7,␣
 ↪label='Método 2')
    plt.title(f'Histograma - Método 2 (N = {n_points})')
    plt.xlabel('Estimativa da Integral')
    plt.ylabel('Frequência')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Função 01

[7]:
```
FUNC_1_LIM_INF = 0
FUNC_1_LIM_SUP = 1
```

[8]:
```
def func_1(x):
    return 1 - x**2
```

Para 100 pontos:

[9]:
```
n_points = 100
```

[10]:
```
first_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

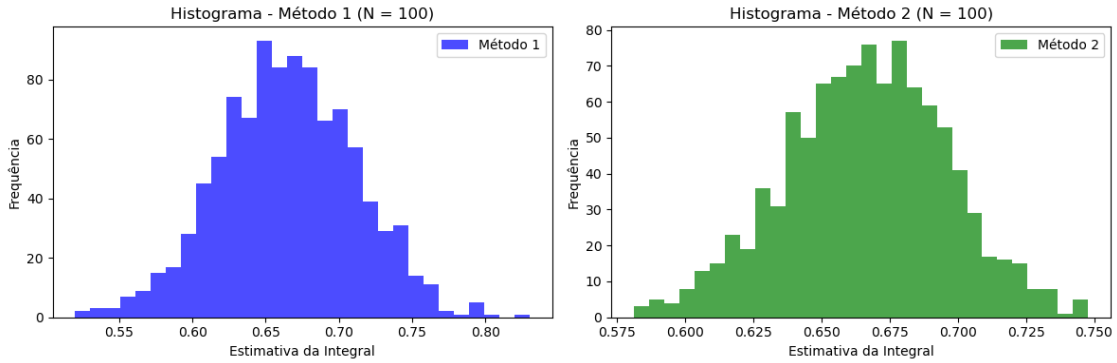
[10]: 0.71

[11]:
```
second_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

[11]: 0.6514143524626452

[12]:
```
estimates_method_1 = get_histogram(func_1, first_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_1, second_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
```

[13]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 1000 pontos:

[14]:
```
n_points = 1000
```

[15]:
```
first_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

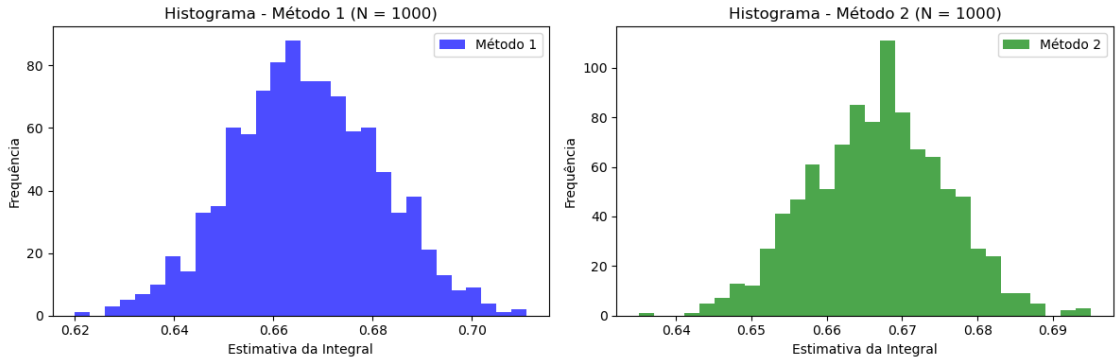
[15]: 0.657

[16]:
```
second_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

[16]: 0.6658789534992553

[17]:
```
estimates_method_1 = get_histogram(func_1, first_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_1, second_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
```

[18]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 10000 pontos:

[19]:
```
n_points = 10000
```

[20]:
```
first_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

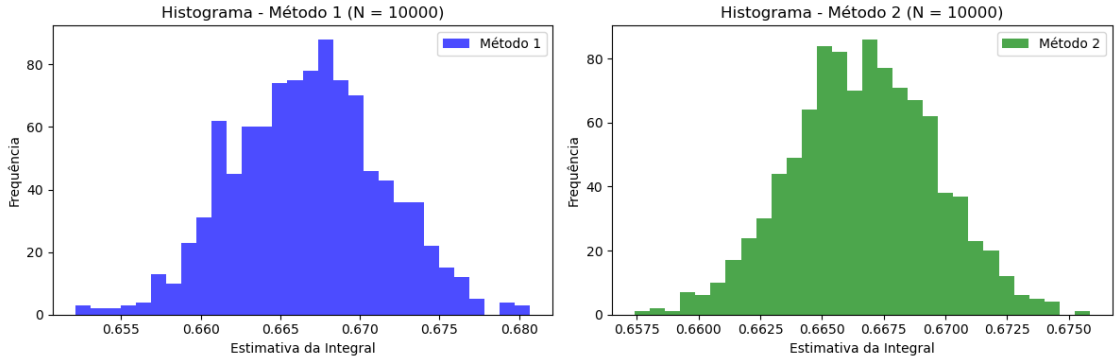
[20]:
```
0.6624
```

[21]:
```
second_method(func_1, FUNC_1_LIM_INF, FUNC_1_LIM_SUP, n_points)
```

[21]:
```
0.6708345280245894
```

[22]:
```
estimates_method_1 = get_histogram(func_1, first_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_1, second_method, FUNC_1_LIM_INF,␣
↪FUNC_1_LIM_SUP, n_points, N_SAMPLES)
```

[23]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Função 02

[24]:
```
FUNC_2_LIM_INF = 0
FUNC_2_LIM_SUP = 1
```

[25]:
```
def func_2(x):
    return np.exp(x)
```

Para 100 pontos:

[26]:
```
n_points = 100
```

[27]:
```
first_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

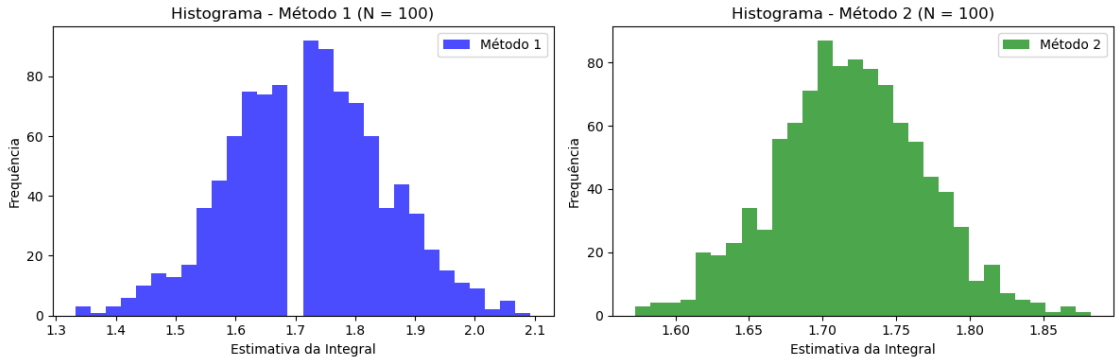
[27]:
```
1.630969097075427
```

[28]:
```
second_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

[28]:
```
1.7348734147127438
```

[29]:
```
estimates_method_1 = get_histogram(func_2, first_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_2, second_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
```

[30]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 1000 pontos:

[31]:
```
n_points = 1000
```

[32]:
```
first_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

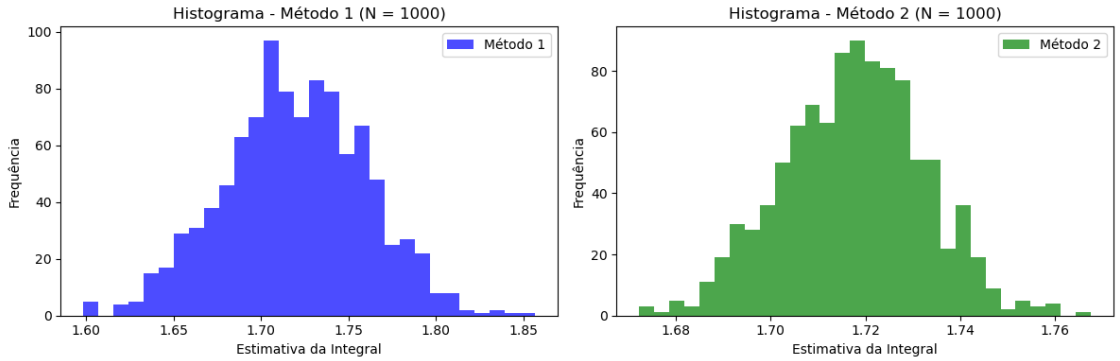
[32]:
```
1.7641649066699203
```

[33]:
```
second_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

[33]:
```
1.7301747560590606
```

[34]:
```
estimates_method_1 = get_histogram(func_2, first_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_2, second_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
```

[35]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 10000 pontos:

[36]:
```
n_points = 10000
```

[37]:
```
first_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

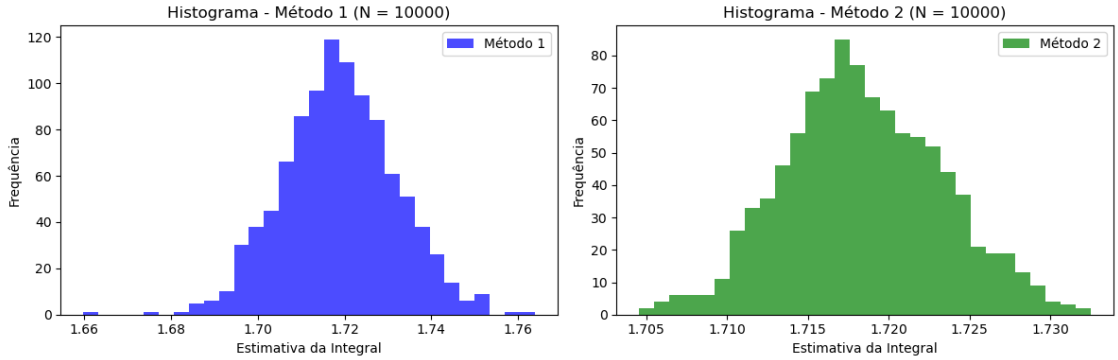
[37]:
```
1.6991979709697491
```

[38]:
```
second_method(func_2, FUNC_2_LIM_INF, FUNC_2_LIM_SUP, n_points)
```

[38]:
```
1.710487301327385
```

[39]:
```
estimates_method_1 = get_histogram(func_2, first_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_2, second_method, FUNC_2_LIM_INF,␣
↪FUNC_2_LIM_SUP, n_points, N_SAMPLES)
```

[40]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Função 03

[41]:
```
FUNC_3_LIM_INF = 0
FUNC_3_LIM_SUP = np.pi
```

[42]:
```
def func_3(x):
    return np.sin(x)**2
```

Para 100 pontos:

[43]:
```
n_points = 100
```

[44]:
```
first_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

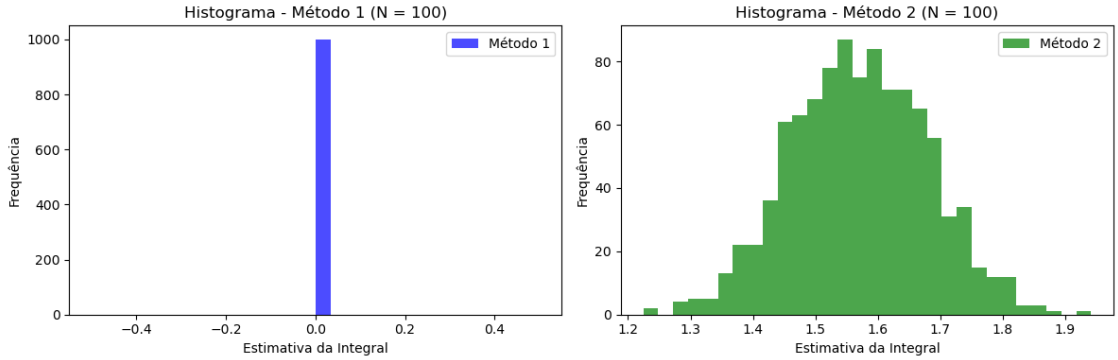
[44]:
```
4.7116343153599164e-32
```

[45]:
```
second_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

[45]:
```
1.4606371624762795
```

[46]:
```
estimates_method_1 = get_histogram(func_3, first_method, FUNC_3_LIM_INF,␣
↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_3, second_method, FUNC_3_LIM_INF,␣
↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
```

[47]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 1000 pontos:

[48]:
```
n_points = 1000
```

[49]:
```
first_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

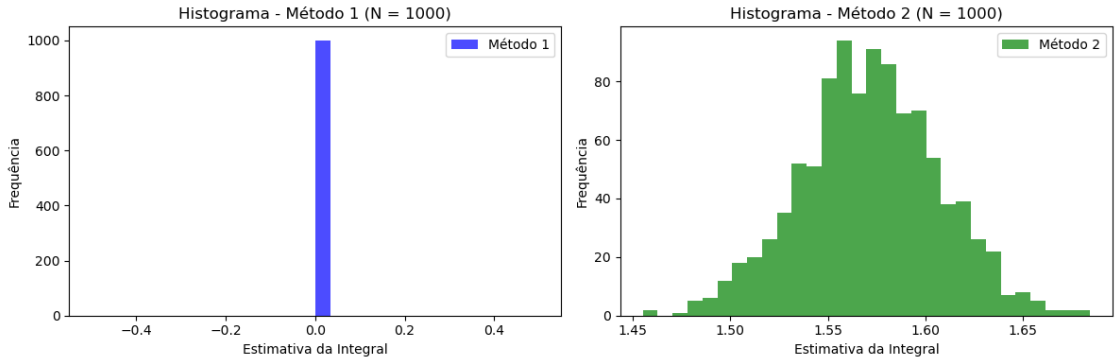
[49]:
```
4.7116343153599164e-32
```

[50]:
```
second_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

[50]:
```
1.5872697023082394
```

[51]:
```
estimates_method_1 = get_histogram(func_3, first_method, FUNC_3_LIM_INF,␣
↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_3, second_method, FUNC_3_LIM_INF,␣
↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
```

[52]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Para 10000 pontos:

[53]:
```
n_points = 10000
```

[54]:
```
first_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

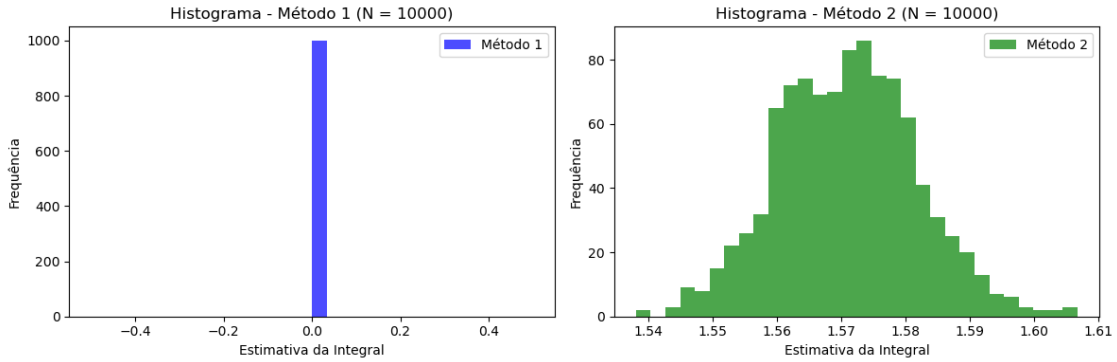
[54]:
```
4.7116343153599164e-32
```

[55]:
```
second_method(func_3, FUNC_3_LIM_INF, FUNC_3_LIM_SUP, n_points)
```

[55]:
```
1.5934887639585515
```

[56]:
```
estimates_method_1 = get_histogram(func_3, first_method, FUNC_3_LIM_INF,␣
 ↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
estimates_method_2 = get_histogram(func_3, second_method, FUNC_3_LIM_INF,␣
 ↪FUNC_3_LIM_SUP, n_points, N_SAMPLES)
```

[57]:
```
plot_histogram(estimates_method_1, estimates_method_2, n_points)
```



Função 04

[58]:
```
FUNC_4_LIM_INF = 0
FUNC_4_LIM_SUP = 1
```

[59]:
```
def func_4(x: list):
    return 1 / ((x[0] + x[1]) * x[2] + (x[3] + x[4]) * x[5] + (x[6] + x[7]) *␣
 ↪x[8])
```

[60]:
```
def second_method_9d(N, func: Callable):
    accumulator = 0
    for _ in range(N):
        accumulator = accumulator + func(np.random.uniform(0, 1, 9))
    return accumulator / N
```

[61]:
```
def run_second_method_9d(n_points):
    sample = np.zeros(N_SAMPLES)
```

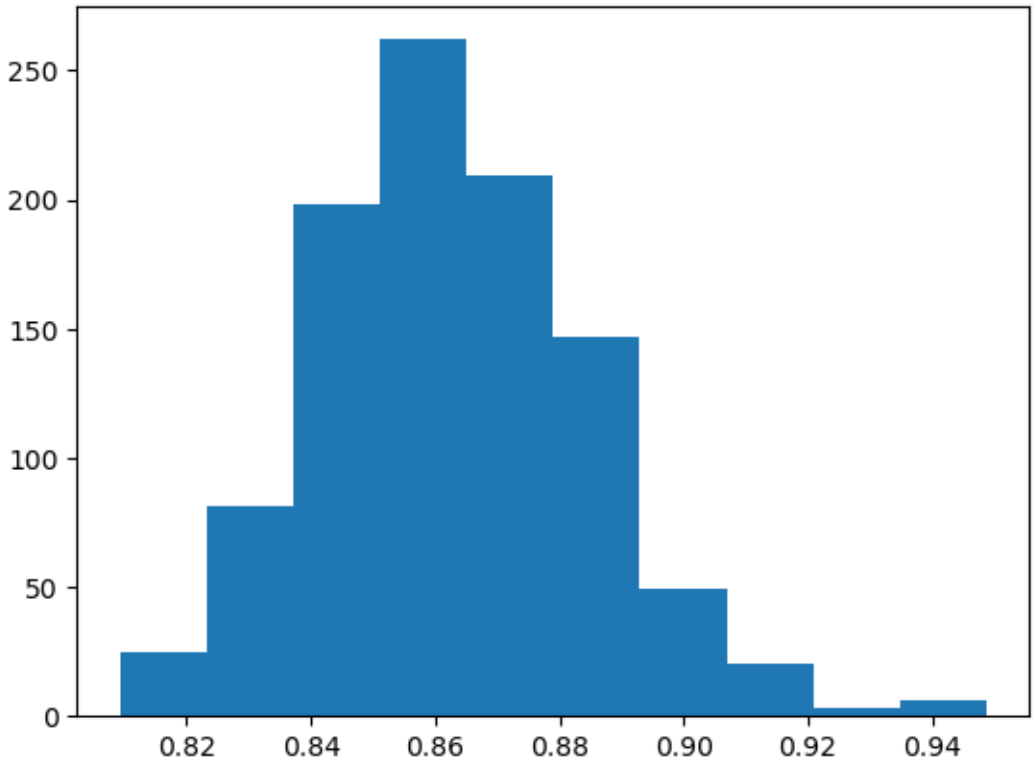
```
    for i in range(N_SAMPLES):
        sample[i] = second_method_9d(n_points, func_4)
    return sample.mean(), sample
```

Para 1000 pontos:

[62]:
```
n_points = 1000
```

[63]:
```
mean, sample = run_second_method_9d(n_points)
```

[64]:
```
plt.hist(sample)
plt.show()
```



[65]:
```
mean
```

[65]: 0.8629080977513331

Conclusões A distribuição dos valores gerados nos histogramas se aproxima da distribuição normal, resultado já esperado, dado o Teorema do Limite Central. Além disso, podemos observar que as médias são próximas aos valores analíticos apresentados.