montecarlo

May 14, 2025

1 Monte Carlo PI

1.1 Roksana Jandura

1.1.1 Funkcje do estymacji liczby PI

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

def estimate_pi(N):
    #Tworzy N losowych liczb w [0, 1] x [0, 1]
    x = np.random.rand(N) #wektor o długości N
    y = np.random.rand(N)

    # Sprawdzamy, czy punkt leży wewnątrz ćwiartki koła: x^2 + y^2 <= 1
    inside = (x**2 + y**2) <= 1.0 #tablic boolowsk

    # Liczymy estymatę Pi z uwzględnieniem, że to 1/4 koła
    pi_estimate = 4.0 * np.sum(inside) / N #suma punktow w cwiartce kola przez␣
↪wszytskie punkty w kwadracie

    #oblicza skumulowaną sumę - czyli ile było trafionych punktow po 1␣
↪losowaniu, 2,3,4,5...N
    cumulative_inside = np.cumsum(inside)
    #oblicza kolejne wartości pi dla N losowań
    running_pi = 4.0 * cumulative_inside / (np.arange(N) + 1)

    #zwracamy wyestymowaną pi, wspolrzedne punktow,
    # tablica informująca które z N punktów trafiły do ćwiartki koła
    #kolejne wartosci pi
    return pi_estimate, x, y, inside,running_pi

def estimate_pi_multiple_runs(N, runs=10):
#przyjmuje liczbe punktow w pojedynczym uruchomieniu, runs czyli ile razy␣
↪powtarzamy losowanie dla danego N
    estimates = []
    #runs razy wykonujemy estimate_pi
    for _ in range(runs):
```

```
        pi_est, _, _, _,_ = estimate_pi(N)
        estimates.append(pi_est) #wyestymowaną liczbe pi doklejamy do listy
    return estimates
```

1.1.2 Funkcje odpowiedzialne za wizualizację (w formie oddzielnych funkcji, aby można było wywołać z różnymi parametrami)

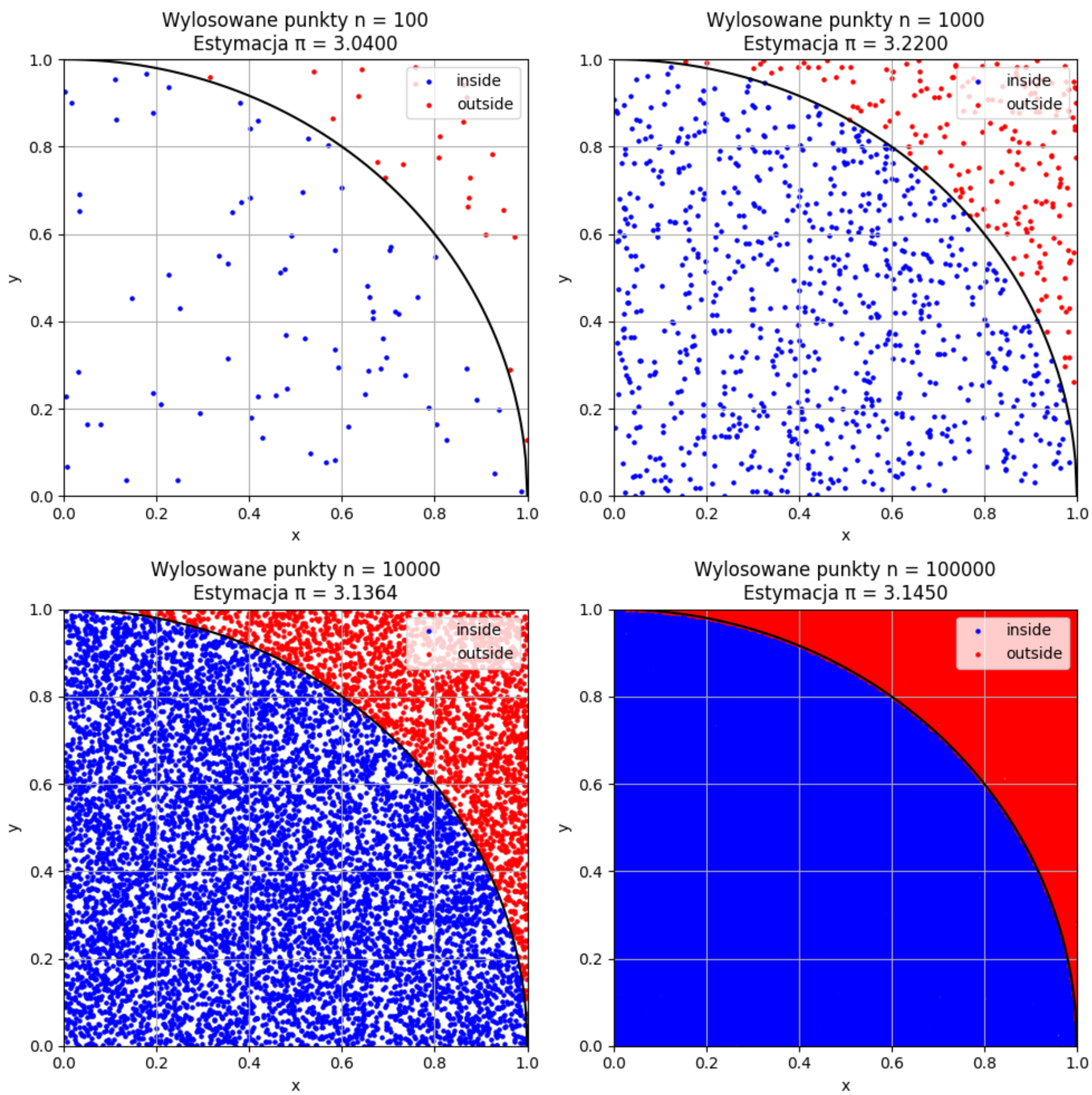
[3]:
```
def plot_four_subplots_for(sample_sizes):

    fig, axes = plt.subplots(2, 2, figsize=(10, 10))
    axes = axes.flatten()

    for i, N in enumerate(sample_sizes):
        pi_est, x, y, inside,_ = estimate_pi(N)
        ax = axes[i]

        ax.scatter(x[inside], y[inside], s=5, c='blue', label='inside')
        ax.scatter(x[~inside], y[~inside], s=5, c='red', label='outside')
        circle = np.linspace(0, 1, 300)
        ax.plot(circle, np.sqrt(1 - circle**2), c='black')
        ax.set_xlim(0, 1)
        ax.set_ylim(0, 1)
        ax.set_title(f"Wylosowane punkty n = {N}\nEstymacja   = {pi_est:.4f}")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.legend(loc='upper right')
        ax.grid(True)

    plt.tight_layout()
    plt.show()

def plot_running_estimate(N):
#pokazuje dążenie estymowanej wartości liczby pi wraz ze wzrostem losowan N

    pi_est, x, y, inside, running_pi = estimate_pi(N)
    plt.figure(figsize=(8,4))
    plt.plot(running_pi, label='Estymata   (bieżąca)')
    plt.axhline(np.pi, color='r', linestyle='--', label='  (referencja)')
    plt.xlabel("Liczba punktów (k)")
    plt.ylabel("Estymowana wartość  ")
    plt.title("Bieżąca estymata   w trakcie narastania liczby losowań")
    plt.legend()
    plt.grid(True)
    plt.show()



def plot_multiple_running_estimates(N=10000, runs=10):
```

```python
    #pokazuje dążeniee estymowanej wartości liczby pi wraz ze wzrostem losowan
    #znajduje sie 'runs' niezależnych lini dążenia

    plt.figure(figsize=(8,5))

    for run_id in range(runs):
        pi_est, x, y, inside,running_pi = estimate_pi(N)
        plt.plot(running_pi, alpha=0.6, label=f'run #{run_id+1}' if run_id == 0␣
 ↪else "")

    plt.axhline(np.pi, color='red', linestyle='--', label='  (prawdziwe)')
    plt.ylim(2, 4)
    plt.title(f"{runs} ścieżek estymacji   przy N={N}")
    plt.xlabel("Liczba punktów (k)")
    plt.ylabel("Estymowana wartość  ")
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_boxplot_estimates(sample_sizes, runs=10):
#wykres pudelkowy
#punkt 3
    all_estimates = []

    #iterujemy przez każdą z wartosci z sample_sizes
    for N in sample_sizes:
        #Wykonujemy runs razy estymację liczby   dla danego N
        estimates = estimate_pi_multiple_runs(N, runs=runs)
        # Dodajemy listę uzyskanych estymacji do all_estimates
        all_estimates.append(estimates)

    plt.figure(figsize=(10,6))
    plt.boxplot(all_estimates, labels=[str(n) for n in sample_sizes])
    plt.axhline(np.pi, color='red', linestyle='--', label='  (referencja)')
    plt.title(f'Porównanie estymat   dla różnych N (każde {runs} powtórzeń)')
    plt.xlabel("Liczba punktów (N)")
    plt.ylabel("Estymowana wartość  ")
    plt.legend()
    plt.grid(True)
    plt.show()




def plot_four_histograms(sample_sizes, runs=50):
```

```python
    # 4 histogramy
    #Każdy  prezentuje rozkład estymacji Pi
    #uzyskanych w 'runs' niezależnych uruchomieniach dla danego N.
    #sample_sizes - lista czterech wartości N (np. [100, 1000, 10000, 100000]).
    #runs - liczba powtórzeń dla każdego N.
    fig, axes = plt.subplots(2, 2, figsize=(10,8))
    axes = axes.flatten()

    for i, N in enumerate(sample_sizes):
        estimates = estimate_pi_multiple_runs(N, runs=runs)
        ax = axes[i]
        ax.hist(estimates, bins=10, color='skyblue', edgecolor='black')
        ax.axvline(np.pi, color='red', linestyle='--', label='  (prawdziwe)')
        ax.set_title(f"N = {N}")
        ax.set_xlabel("Przybliżona wartość  ")
        ax.set_ylabel("Częstość występowania")
        ax.legend()
        ax.grid(True)

    fig.suptitle(f"Rozkład estymowanej wartości   dla różnej liczby punktów dla␣
 ↪runs= {runs}", fontsize=14)
    plt.tight_layout()
    plt.show()
```

Ten wykres przedstawia losowo wygenerowane punkty w pierwszej ćwiartce układu współrzędnych w celu przybliżenia liczby PI metodą Monte Carlo.

[11]:
```python
sample_sizes = [100, 1000, 10000, 100000]

plot_four_subplots_for(sample_sizes)
```

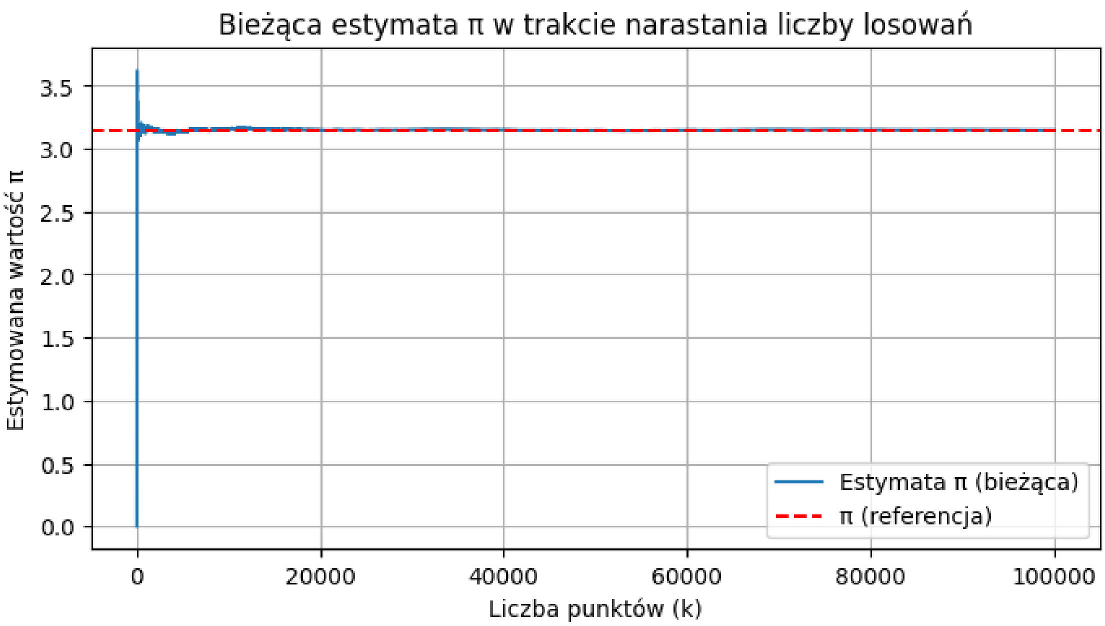


Wnioski:

Im więcej punktów, tym bardziej widoczny i dokładny staje się kształt ćwiartki koła.

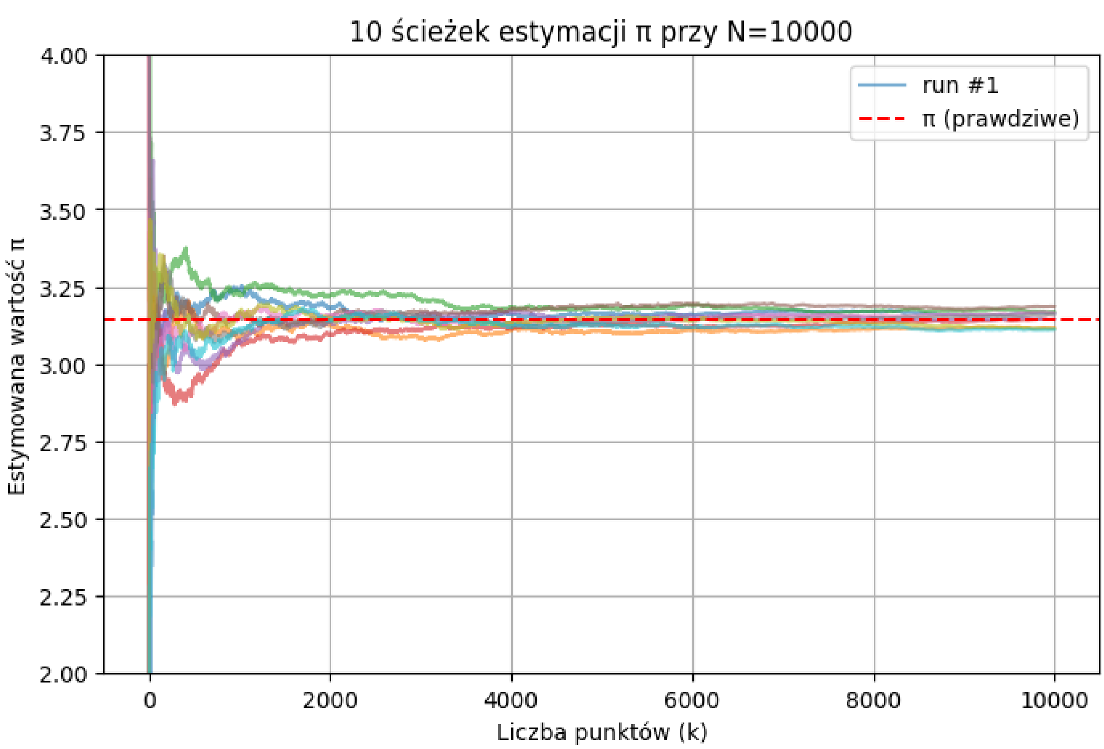
Wykres przedstawia “dążenie” chwilowej estymowanej wartości PI wraz ze wzrostem ilości losowań

[12]:
```
plot_running_estimate(sample_sizes[3])
```



Ten wykres przedstawia 10 niezależnych ścieżek estymacji liczby PI dla N=10000 punktów w każdej próbie. Każda linia pokazuje, jak w danej próbie estymowana wartość PI zbiega do wartości rzeczywistej.

[13]:
```
plot_multiple_running_estimates()
```

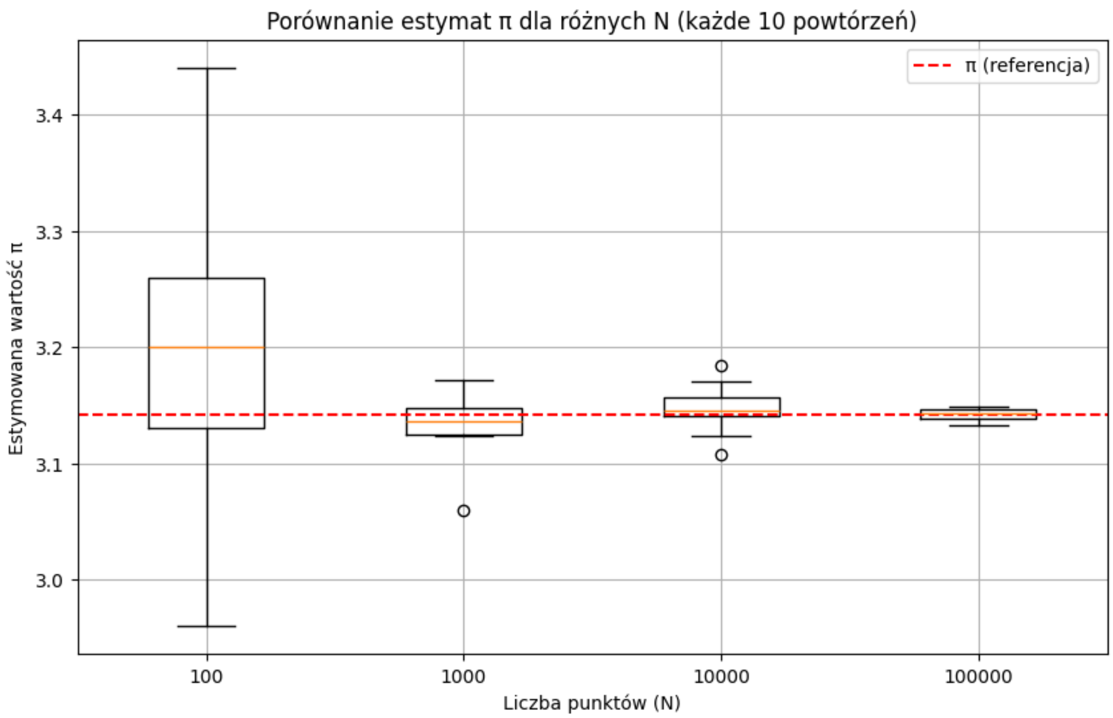


Wnioski:

Wszystkie ścieżki, mimo początkowych wahań, stabilizują się wokół prawdziwej wartości, co potwierdza skuteczność metody Monte Carlo.

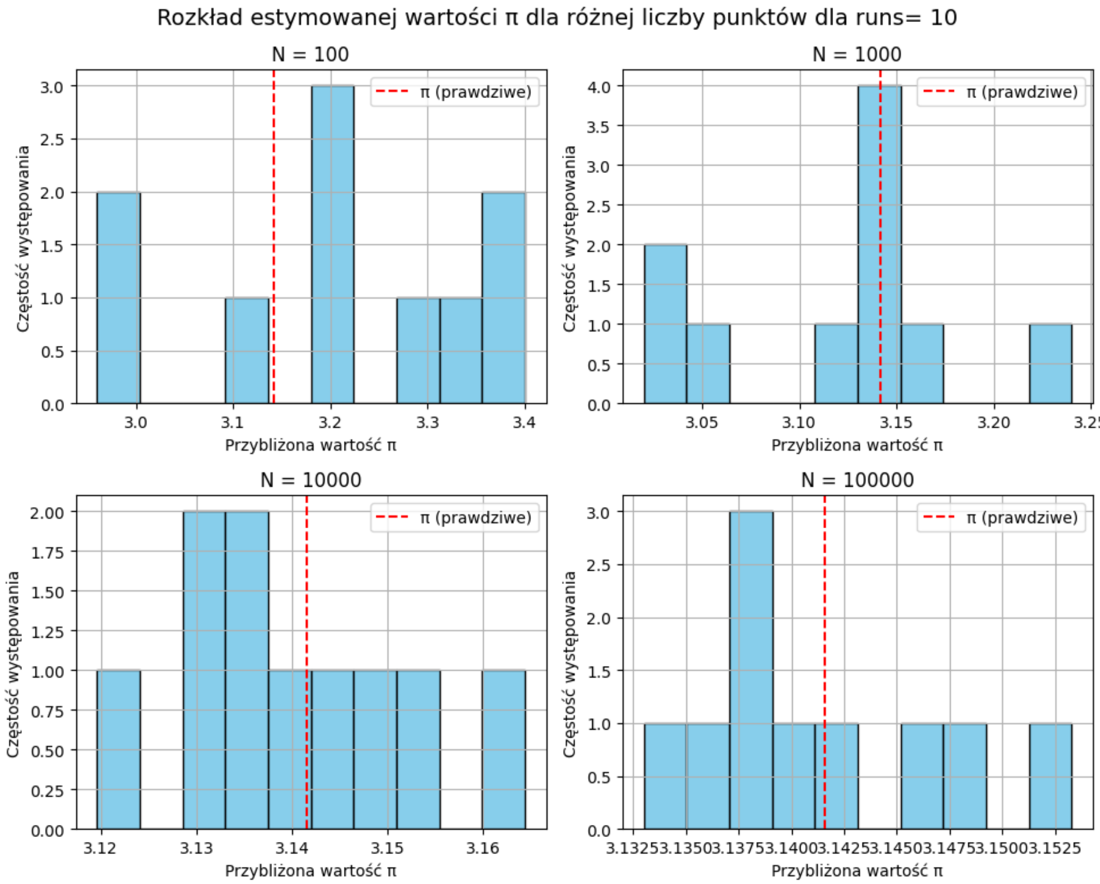
Wykres pudełkowy przedstawia rozrzut estymacji liczby PI dla różnych liczności próby N, gdzie dla każdej wartości N wykonano 10 powtórzeń.

[14]:
```
plot_boxplot_estimates(sample_sizes, runs=10)
```

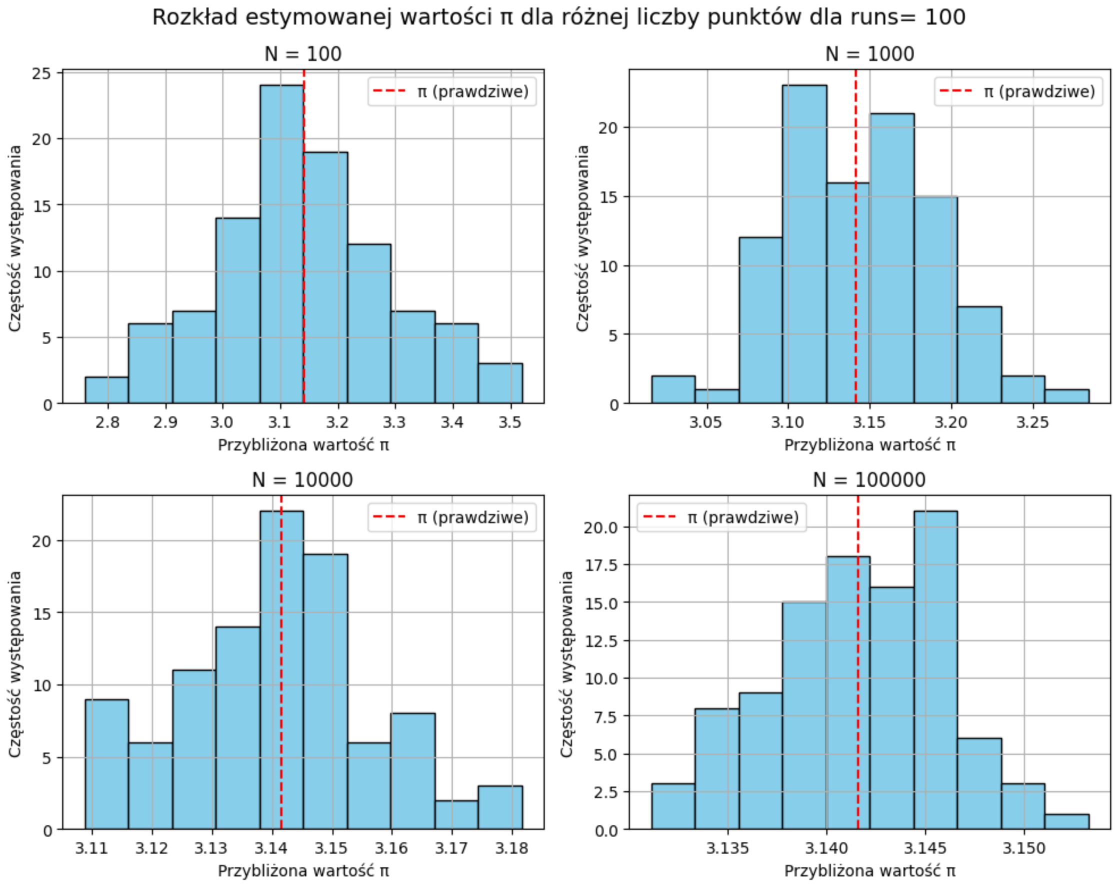


Powyższy wykres potwierdza poprawę jakości estymacji wraz ze wzrostem N.

[8]:
```
plot_four_histograms(sample_sizes, runs=10)
```




```
[9]: plot_four_histograms(sample_sizes, runs=100)
```



Im większa liczba punktów N, tym dokładniejsze i bardziej stabilne estymacje PI. Im większa liczba powtórzeń runs, tym lepiej widoczny i gładki rozkład, co zwiększa wiarygodność statystyczną.

```
[ ]:
```