Użycie metody monte carlo do estymacji liczby pi

Autor projektu: Adam Lewiński, IiAD III rok, GRUPA 2

Importowanie potrzebnych bibliotek.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Funkcje napisane przeze mnie, aby ułatwić sobie dalszą pracę.
Funkcje samplePoints i createPoints odpowiedzialne są za wygenerowanie punktów.
Funkcja segregatePoints rozdziela punkty, na te które leżą w obrębie koła oraz poza nim.
Funkcja estimatePi oblicza estymowaną wartość liczby pi, na podstawie dokonanego podziału punktów.

In [2]:

```
def samplePoints(generator, series, points):
    result = []
    for i in range(series):
        df = pd.DataFrame(generator.random((points,2)), columns = ["X","Y"])
        result.append(df)
    return result

def createPoints(generator, series):
    result1 = samplePoints(generator, series, 10)
    result2 = samplePoints(generator, series, 100)
    result3 = samplePoints(generator, series, 1000)
    result4 = samplePoints(generator, series, 10000)

    return [result1, result2, result3, result4]

def segregatePoints(points):
    result = []
    for elem in points:
        points = []
        for df in elem:
            circle = []
            square = []
            for index, row in df.iterrows():
                if row["X"] ** 2 + row["Y"] ** 2 <= 1:
                    circle.append([row["X"],row["Y"]])
                else:
                    square.append([row["X"],row["Y"]])
            series = [circle,square]
            points.append(series)
        result.append(points)
    return result
```

In [3]:

```
def estimatePi(segregatedPoints):
    result = []
    for elem in segregatedPoints:
        pi = []
        for serie in elem:
            pi.append( len(serie[0])/(len(serie[0]) + len(serie[1]))*4 )
        result.append(pi)
    return result
```

Poniższa funkcja odpowiedzialna jest za wykonanie odpowiednich czynności, które prowadzą do możliwości wizualizacji danych w postaci boxplot, dla ustalonego generatora.

In [4]:

```
def procedureForGenerator(generator, series):
    points = createPoints(generator,series)
    segregatedPoints = segregatePoints(points)
    data = estimatePi(segregatedPoints)
    return data
```

Funkcja singlePlot odpowiedzialna jest za wykonanie pojedyńczego wykresu.
Funkcja createPlot kolejno dla czterech generatorów generuje wyniki oraz je wizualizuje.

In [5]:

```
def singlePlot(ax, data, generatorName, series):
    linex = [0,1,2,3,4,5]
    liney = [np.pi, np.pi, np.pi, np.pi, np.pi, np.pi ]

    ax.set_title(f"Estymacja liczby pi przy 10,100,1000 i 10000 punktach - {gener
    ax.set_xlabel("1*10^x generowanych punktów")
    ax.set_ylabel(f"Wartość estymacji liczby pi dla {series} serii")
    ax.set_ylim([np.pi-1/2*np.pi,np.pi+1/2*np.pi])
    ax.boxplot(data)
    ax.plot(linex,liney,c = "blue", linewidth=1, alpha=0.5)

    return ax
```

In [ ]:

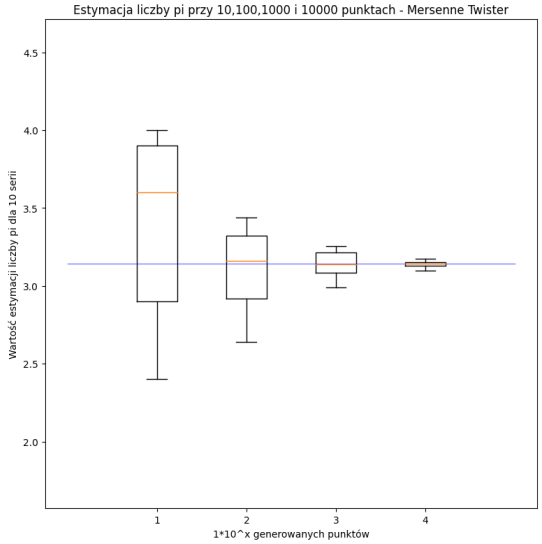
```
def createBoxPlots(seed):
    series = 10

    generator = np.random.Generator(np.random.MT19937(seed))
    data1 = procedureForGenerator(generator, series)

    generator = np.random.Generator(np.random.PCG64(seed))
    data2 = procedureForGenerator(generator, series)

    generator = np.random.Generator(np.random.Philox(seed))
    data3 = procedureForGenerator(generator, series)

    generator = np.random.Generator(np.random.SFC64(seed))
    data4 = procedureForGenerator(generator, series)


    fig,ax = plt.subplots(2,2, figsize =(20, 20))
    fig.suptitle(f"Seed: {seed}")
    ax[0,0] = singlePlot(ax[0,0], data1, "Mersenne Twister", 10)
    ax[0,1] = singlePlot(ax[0,1], data2, "Permuted Congruential Generator", 10)
    ax[1,0] = singlePlot(ax[1,0], data3, "Philox Counter-based RNG", 10)
    ax[1,1] = singlePlot(ax[1,1], data4, "SFC64 Small Fast Chaotic PRNG", 10)
```

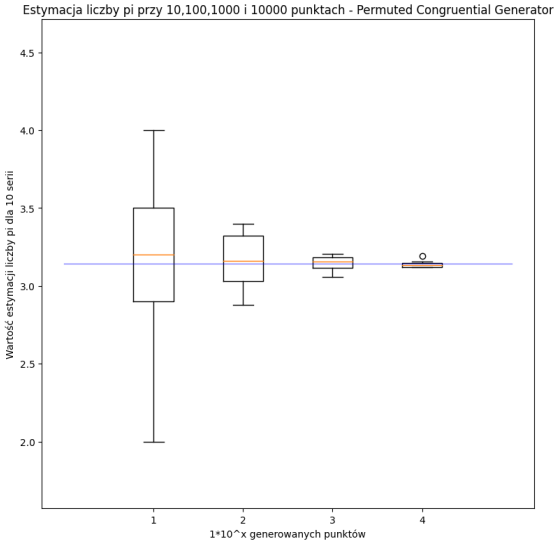
Wywołanie funkcji w celu otrzymania rezultatów.

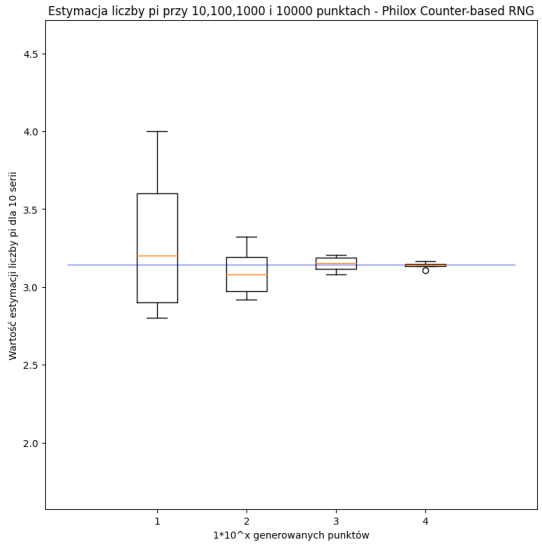
In [6]:

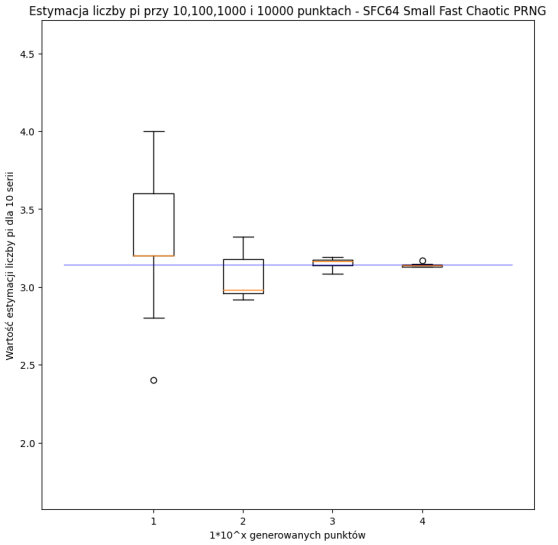
```
createBoxPlots(12345)
```

Seed: 12345





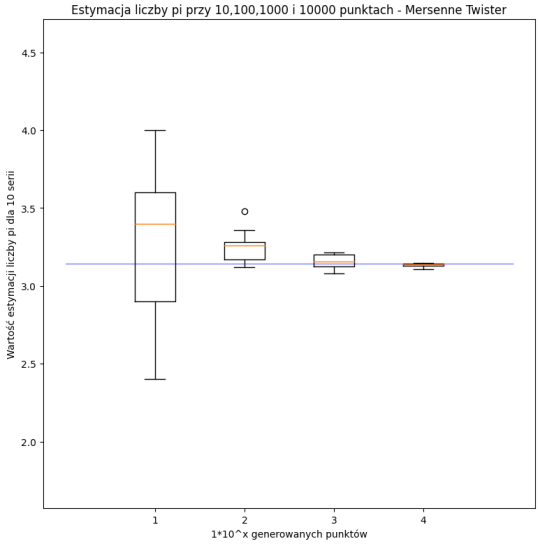


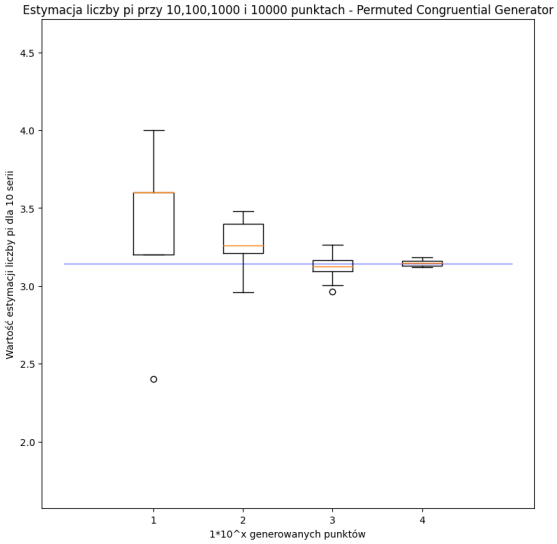


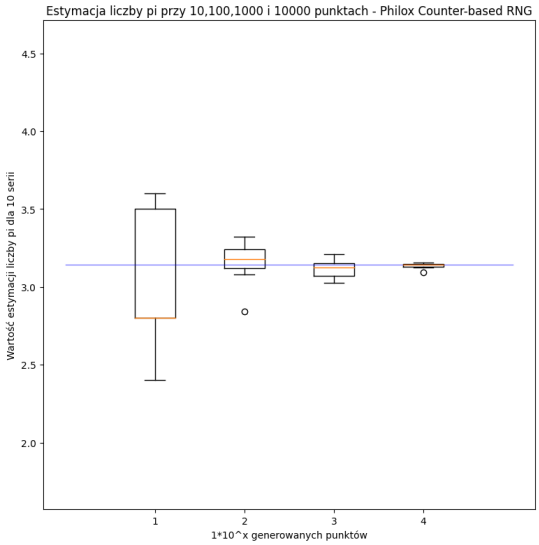
In [7]:

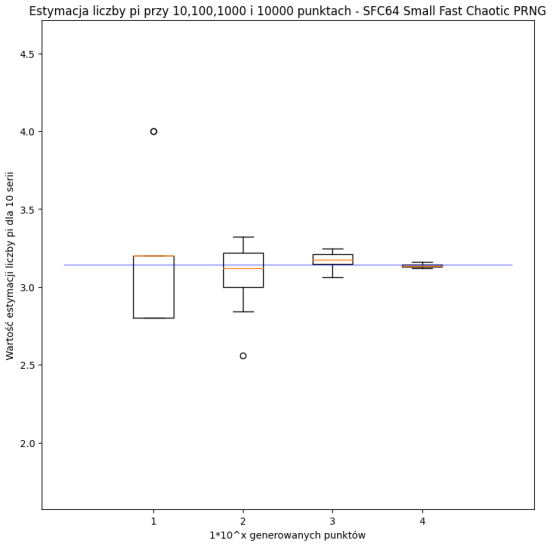
```
createBoxPlots(937162211)
```

Seed: 937162211





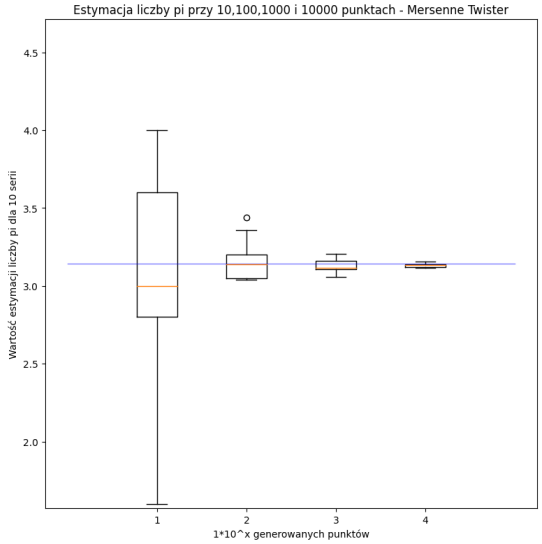


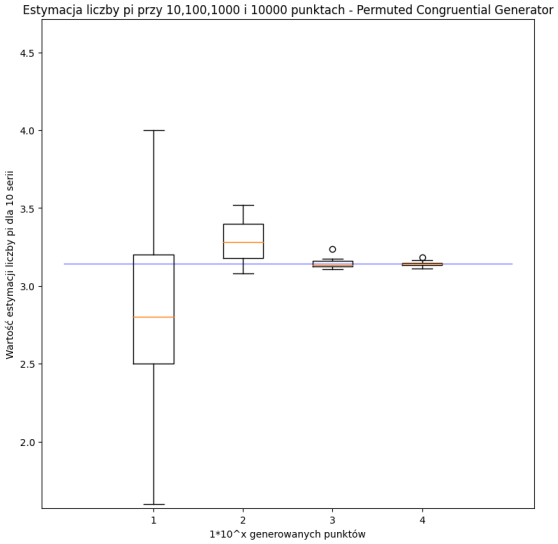


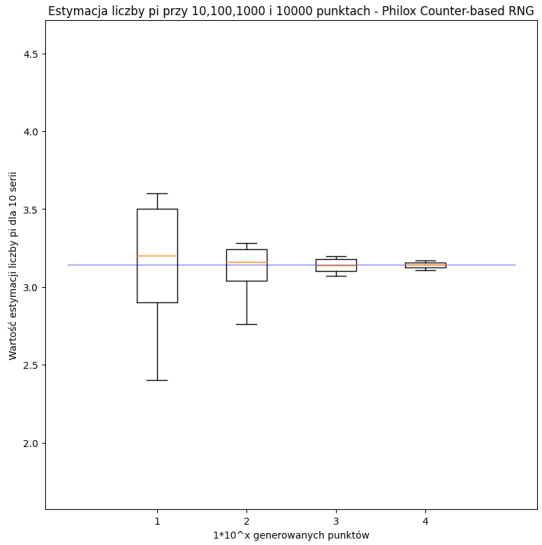
In [8]:

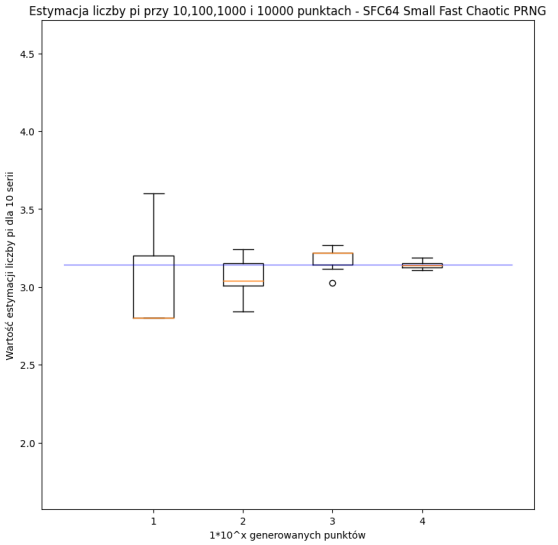
```
createBoxPlots(0)
```

Seed: 0









Poniższe funkcje to lekko zmodyfikowane wersje powyższych oraz jedna nowa procedura.
Procedura ma za zadanie obliczyć estymację liczby pi po wylosowaniu i punktów, zwiększyć i o 1 i powtórzyć proces aż do otrzymania wyniku dla zadanej liczby punktów.

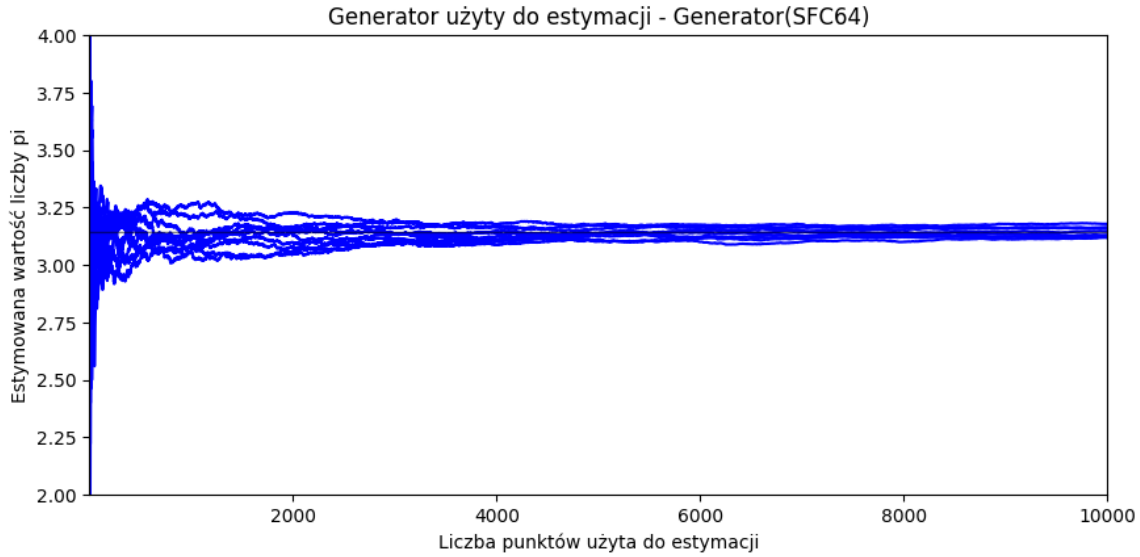
In [140]:

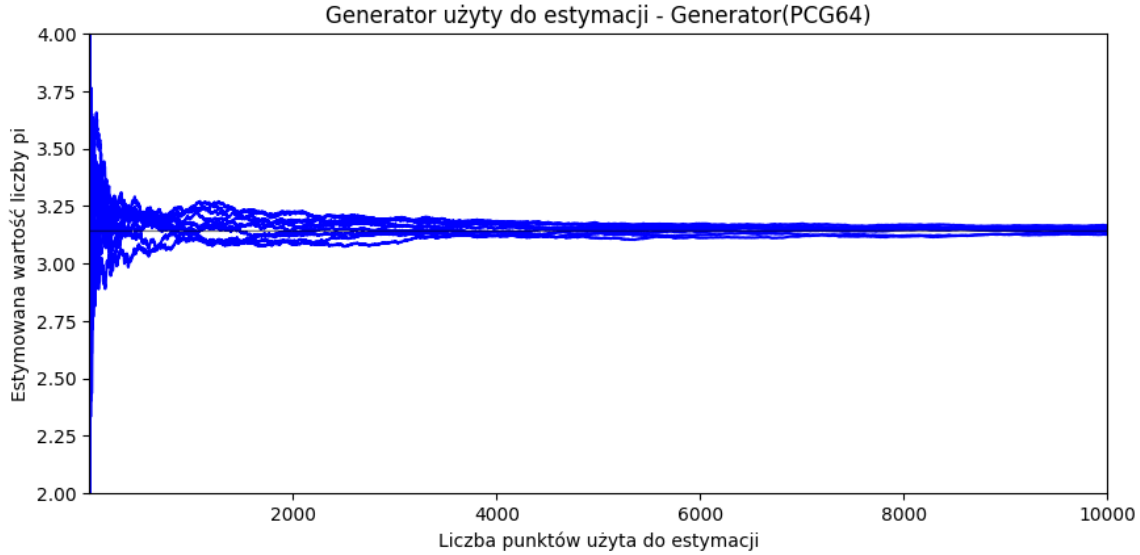
```
def createPoints2(generator, series, points):
    result = samplePoints(generator, series, points)
    return [result,]

def segregatePoints2(pointsInput):
    result = []
    circle = [[] for i in range(len(pointsInput[0]))]
    square = [[] for i in range(len(pointsInput[0]))]

    for i in range(len(pointsInput[0][0])):
        for j in range(len(pointsInput[0])):
            df = pointsInput[0][j]
            row = df.iloc[i,:]
            if row["X"] ** 2 + row["Y"] ** 2 <= 1:
                circle[j].append([row["X"],row["Y"]])
            else:
                square[j].append([row["X"],row["Y"]])
            series = [circle[j],square[j]]
            try:
                result[j] = [series]
            except:
                result.append([series])
        yield result

def procedureOfEstimatingPi(generator, series, points, rangePoints):
    data = createPoints2(generator, series, points)
    pi = []
    mygenerator = segregatePoints2(data)
    for i in rangePoints:
        segregatedPoints = next(mygenerator)
        pi.append(estimatePi(segregatedPoints))
    return pi
```

Do wizualizacji estymacji liczby pi w trakcie dokładania punktów wybrałem dwa generatory - Small Fast Chaotic PRNG oraz Permuted Congruential Generator.

In [142]:

```
# Dwa najciekawsze generatory moim zdaniem SFC64 i PCG64
series = 10
points = 10000
seed = 0

generators = []
generators.append(np.random.Generator(np.random.SFC64(seed)))
generators.append(np.random.Generator(np.random.PCG64(seed)))

linex = np.linspace(0,points,points)
liney = np.linspace(np.pi, np.pi, points)

fig,ax = plt.subplots(len(generators),1,figsize=(10,10))

for k in range(len(generators)):
    data = procedureOfEstimatingPi(generators[k], series, points, range(1,points+
    for i in range(0,series):
        y = []
        x = np.linspace(1,points, points)
        for elem in data:
            y.append(elem[i])
        ax[k].plot(x,y, c="blue")
    ax[k].set_ylim([2,4])
    ax[k].set_xlim([1,points])
    ax[k].set_xlabel("Liczba punktów użyta do estymacji")
    ax[k].set_ylabel("Estymowana wartość liczby pi")
    ax[k].set_title(f"Generator użyty do estymacji - {generators[k]}")
    ax[k].plot(linex,liney,c = "black", linewidth=1, alpha=0.5)
```





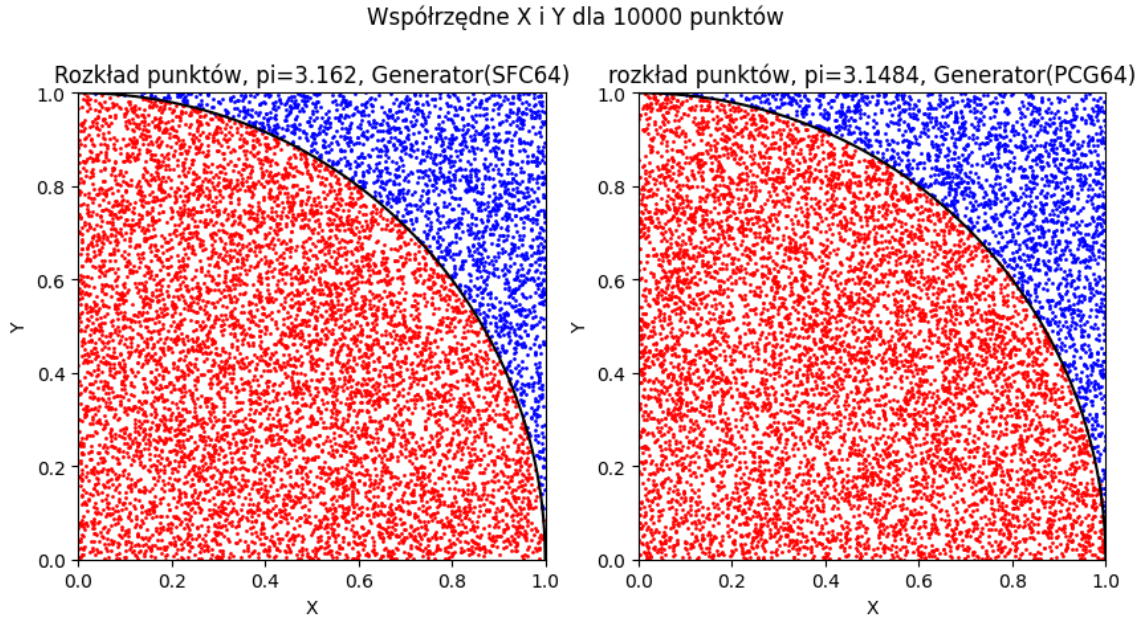
Zwizualizowałem także jak wyglądają takie losowe rozkłady punktów w przestrzenii X i Y.

In [120]:

```
series = 1
pointsNumber = 10000
points1 = createPoints2(generators[0], series, pointsNumber)
points2 = createPoints2(generators[1], series, pointsNumber)
points = [points1[0], points2[0]]
data = segregatePoints(points)

pi = estimatePi(data)

x = np.linspace(0,1,100000)
circle = lambda x: np.sqrt(abs(x**2 - 1))
y = circle(x)

x1c = [x[0] for x in data[0][0][0]]
y1c = [y[1] for y in data[0][0][0]]

x1s = [x[0] for x in data[0][0][1]]
y1s = [y[1] for y in data[0][0][1]]

x2c = [x[0] for x in data[1][0][0]]
y2c = [y[1] for y in data[1][0][0]]

x2s = [x[0] for x in data[1][0][1]]
y2s = [y[1] for y in data[1][0][1]]

fig, ax = plt.subplots(1,2, figsize = (10,5))

fig.suptitle(f"Współrzędne X i Y dla {pointsNumber} punktów")
msize = 1

ax[0].axis("square")
ax[0].set_xlim([0,1])
ax[0].set_ylim([0,1])
ax[0].set_xlabel("X")
ax[0].set_ylabel("Y")
ax[0].set_title(f"Rozkład punktów, pi={pi[0][0]}, {generators[0]}")
ax[0].plot(x,y,c="black")
ax[0].scatter(x1c,y1c,c="red", s = msize)
ax[0].scatter(x1s,y1s,c="blue", s = msize)

ax[1].axis("square")
ax[1].set_xlim([0,1])
ax[1].set_ylim([0,1])
ax[1].set_xlabel("X")
ax[1].set_ylabel("Y")
ax[1].set_title(f"rozkład punktów, pi={pi[1][0]}, {generators[1]}")
ax[1].plot(x,y,c="black")
ax[1].scatter(x2c,y2c,c="red", s = msize)
ax[1].scatter(x2s,y2s,c="blue", s = msize)
```

Out[120]:

```
<matplotlib.collections.PathCollection at 0x7f6ece3b34f0>
```



Wnioski: wraz ze wzrostem liczby punktów, estymacja liczby pi jest dokładniejsza. Jej [estymacji] przebieg zależy od użytego generatora oraz jego ziarna. Ocena wizualna wyników pozwala stwierdzić, że najmniejszą wariancją cechują się wyniki z generatora SFC64 Small Fast Chaotic PRNG, są najbardziej do siebie zbliżone, natomiast niekoniecznie dobrze przybliżają liczbę pi. Uważam, że dla dużej liczby losowanych punktów najlepszy jest Permuted Congruential Generator. Seedem, który dobrze sprawdził się dla tych dwóch generatorów było 0.