Metody Monte Carlo

Laboratorium 4

Zadanie 4

Kod (Python)

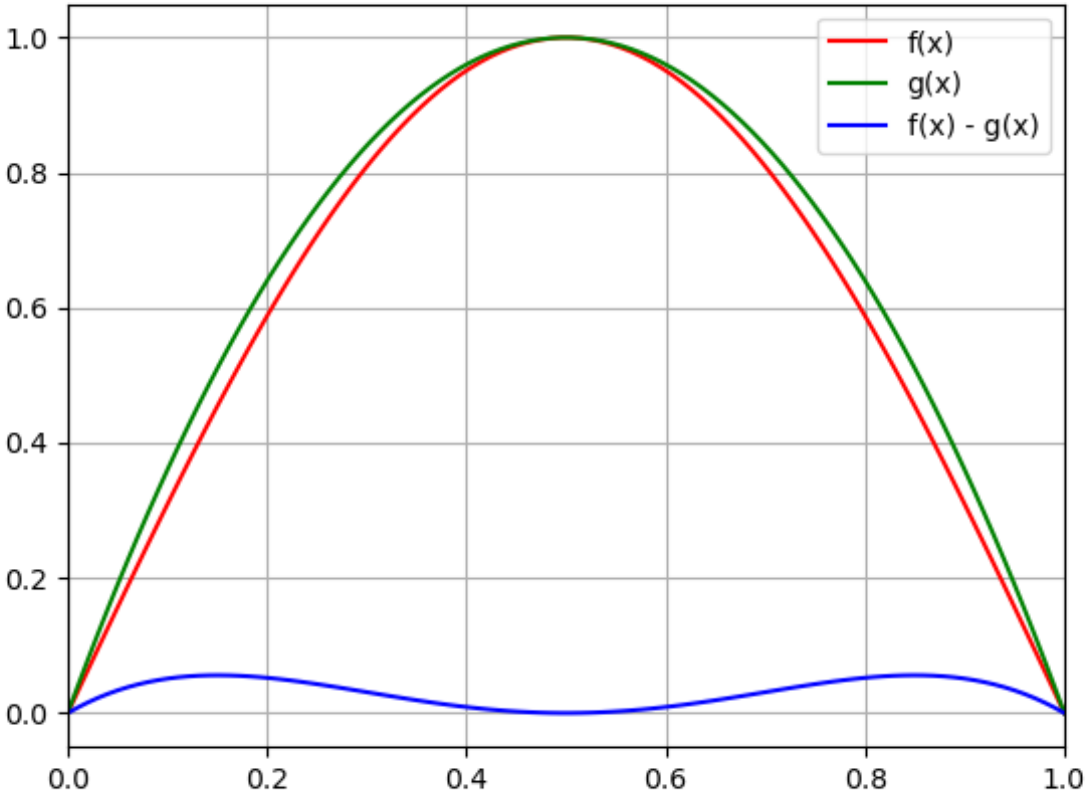
```python
from matplotlib import pyplot as plt
from math import sin, sqrt
import numpy as np

PI = 3.141592653589793


def f(x):
  return sin(PI * x)


def g(x):
  return (-4) * x * (x - 1)


def expectation(array, n):
  probability = 1 / n
  sum = 0
  for i in range(0, n):
    sum += (array[i] * probability)

  return float(sum)


def variance(array):
  mean = sum(array) / len(array)
  res = sum((i - mean)**2 for i in array) / len(array)
  return res


def covariance(f_samples, g_samples, n):
  f_g_sum = [a * b for a, b in zip(f_samples, g_samples)]
  return (1 / n) * sum(f_g_sum) - (1 / n**2) * sum(f_samples) * sum(g_samples)


# http://www.if.pwr.edu.pl/~pater/DANE/niepewnoci_pomiarw.html
def uncertainty(x, x_dash):
  n = len(x)
  return sqrt(sum((xi - x_dash)**2 for xi in x) / (n * (n - 1)))


def map(array, index):
  return [a[index] for a in array]


def main():
  N = 100

  range_01_samples = np.linspace(0, 1, N)
  g_samples = [g(x) for x in range_01_samples]
  f_samples = [f(x) for x in range_01_samples]

  # 1 Wyswietlic na jednym wykresie funckje f(x) oraz g(x) oraz ich roznice w rzedziale x[0, 1]
  f_g_substract = [abs(a - b) for a, b in zip(f_samples, g_samples)]

  plt.figure()
  plt.xlim([0, 1])
  plt.plot(range_01_samples, f_samples, 'r-', label='f(x)')
  plt.plot(range_01_samples, g_samples, 'g-', label='g(x)')
  plt.plot(range_01_samples, f_g_substract, 'b-', label='f(x) - g(x)')
  plt.grid()
  plt.legend()
  # plt.show()

  # Obliczyc analitycznie wartosc calki I
```

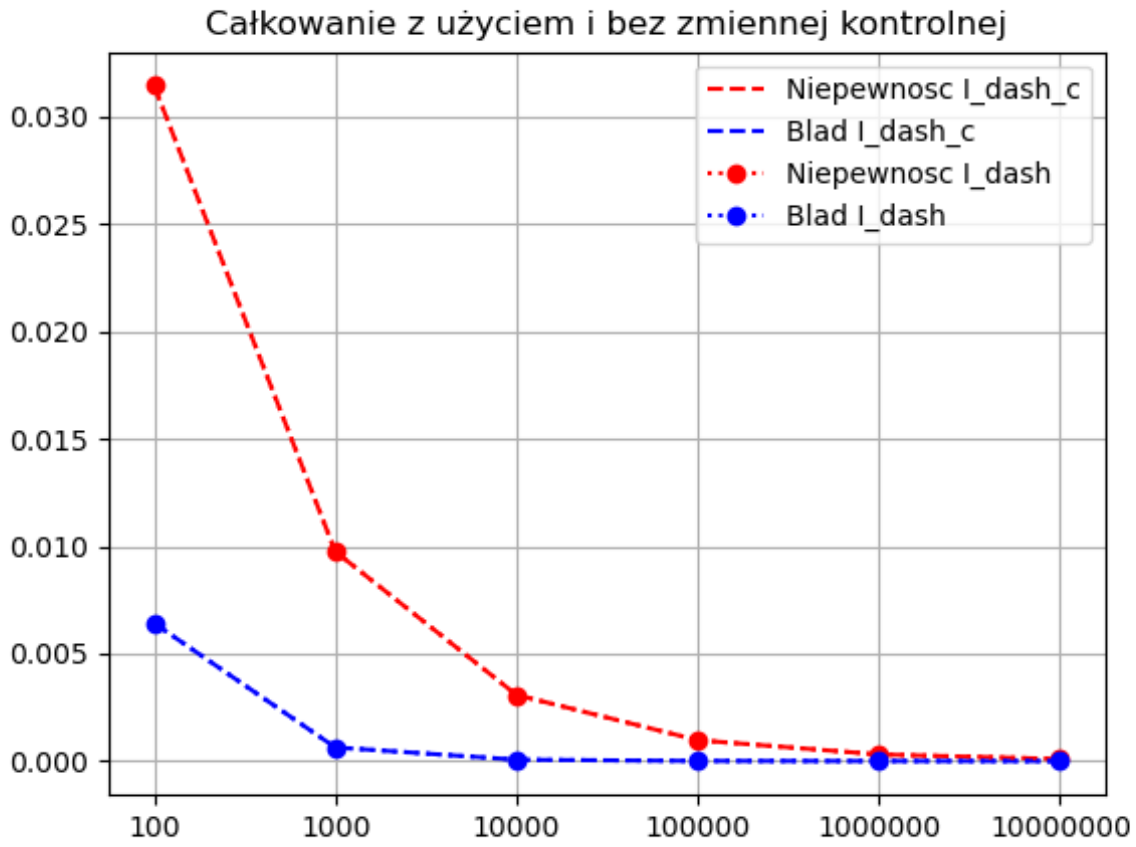
```python
  # https://www.wolframalpha.com/input/?i=integral+sin%CF%80x+from+0+to+1
  # I = 2/π = 0.636619...
  I = 2 / PI

  # Oszacowac wartosc tej calki metoda podstawowa Monte Carlo
  # Z zastosowaniem zmienniej kontrolnej g(x)
  # Bez zmiennej kontrolnej

  I_dash_c_results = []
  I_dash_results = []

  for n in [N * 10**x for x in range(0, 6)]:
    range_samples = np.linspace(0, 1, n)
    f_samples = [f(x) for x in range_samples]
    g_samples = [g(x) for x in range_samples]
    g_variance = variance(g_samples)
    g_exp_dash = expectation(g_samples, n)

    # 1
    f_exp = expectation(f_samples, n)
    # g_exp = expectation(g_samples, n)
    cov = covariance(f_samples, g_samples, n)

    # 2
    c = cov / g_variance

    # 3
    g_gdash_exp = expectation([sample - g_exp_dash for sample in g_samples], n)
    I_dash_c = f_exp - c * g_gdash_exp
    I_dash = f_exp

    # log
    I_dash_c_uncertainty = uncertainty(f_samples, I_dash_c)
    I_dash_c_delta = abs(I_dash_c - I)
    I_dash_uncertainty = uncertainty(f_samples, I_dash)
    I_dash_delta = abs(I_dash - I)

    print(f'N = {n}')
    print(
        f'I w/o correction | Estimate = {I_dash:0.6f} | Uncertainty = {I_dash_uncertainty:0.6f} |
Delta = {I_dash_delta:0.6f}'
    )
    print(
        f'I w/w correction | Estimate = {I_dash_c:0.6f} | Uncertainty = {I_dash_c_uncertainty:0.6f}
| Delta = {I_dash_c_delta:0.6f}'
    )

    I_dash_c_results.append([n, I_dash_c, I_dash_c_uncertainty, I_dash_c_delta])
    I_dash_results.append([n, I_dash, I_dash_uncertainty, I_dash_delta])

  # Przedstawic na wspolnym wykresie zaleznosc niepewnosci i bledu od licznosci proby dla obu
przypadkow
  x_ticks = range(len(I_dash_c_results))

  plt.figure()
  plt.xticks(x_ticks, map(I_dash_c_results, 0))
  plt.plot(x_ticks,
           map(I_dash_c_results, 2),
           'r--',
           label='Niepewnosc I_dash_c')
  plt.plot(x_ticks, map(I_dash_c_results, 3), 'b--', label='Blad I_dash_c')
  plt.plot(x_ticks, map(I_dash_results, 2), 'o:r', label='Niepewnosc I_dash')
  plt.plot(x_ticks, map(I_dash_results, 3), 'o:b', label='Blad I_dash')
  plt.grid()
  plt.title('Całkowanie z użyciem i bez zmiennej kontrolnej')
  plt.legend()

  plt.show()


if __name__ == '__main__':
  main()
```

Wyniki

Output konsoli:

```
N = 100
I w/o correction | Estimate = 0.630201 | Uncertainty = 0.031438 | Delta = 0.006419
I w/w correction | Estimate = 0.630201 | Uncertainty = 0.031438 | Delta = 0.006419
N = 1000
I w/o correction | Estimate = 0.635983 | Uncertainty = 0.009753 | Delta = 0.000637
I w/w correction | Estimate = 0.635983 | Uncertainty = 0.009753 | Delta = 0.000637
N = 10000
I w/o correction | Estimate = 0.636556 | Uncertainty = 0.003078 | Delta = 0.000064
I w/w correction | Estimate = 0.636556 | Uncertainty = 0.003078 | Delta = 0.000064
N = 100000
I w/o correction | Estimate = 0.636613 | Uncertainty = 0.000973 | Delta = 0.000006
I w/w correction | Estimate = 0.636613 | Uncertainty = 0.000973 | Delta = 0.000006
N = 1000000
I w/o correction | Estimate = 0.636619 | Uncertainty = 0.000308 | Delta = 0.000001
I w/w correction | Estimate = 0.636619 | Uncertainty = 0.000308 | Delta = 0.000001
N = 10000000
I w/o correction | Estimate = 0.636620 | Uncertainty = 0.000097 | Delta = 0.000000
I w/w correction | Estimate = 0.636620 | Uncertainty = 0.000097 | Delta = 0.000000
```



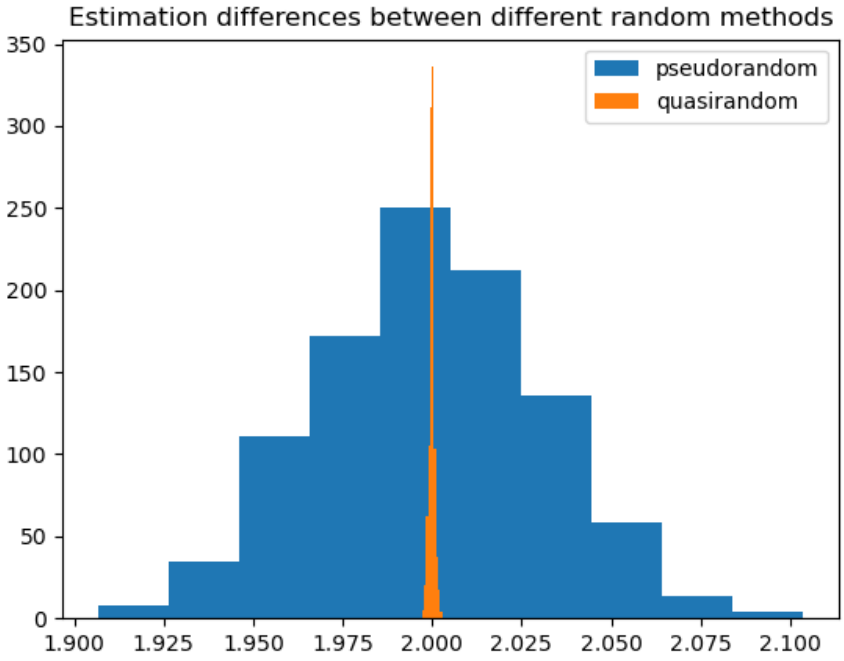


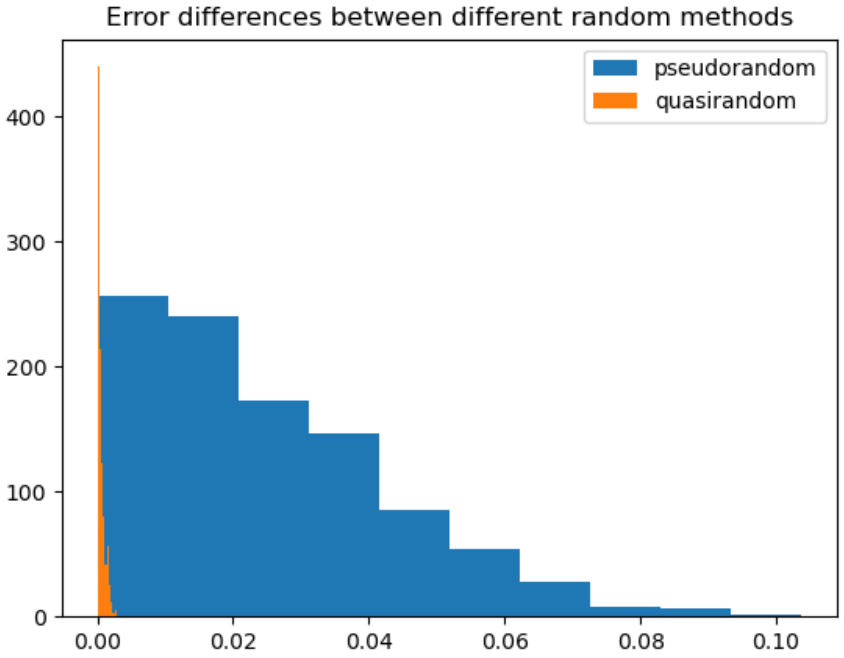
Oczekiwane zachowanie obydwu metod powinno pozwolić na co raz dokładniejszy wynik całkowania, proporcjonalny do wzrostu ilości iteracji, określoną zmienną N. Trend ten utrzymuje się dla obydwu metod i daje praktycznie te same wyniki.
Jest to raczej efekt oczekiwany, niemniej jednak może on także wynikać z niechcianego błędu w obliczeniach.

Zadanie 7
Kod (Python)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import qmc

PI = 3.141592653589793


def pseudorandom(n):
  a = 0
  x = np.random.uniform(a, PI, size=n)
  est = (PI - a) * np.mean(np.sin(x))
  return est, np.abs(est - 2)


def quasirandom(n):
  a = 0
  sampler = qmc.Sobol(d=1, seed=None)
  x = sampler.random(n).T.squeeze() * PI
  est = (PI - a) * np.mean(np.sin(x))
  return est, np.abs(est - 2)


def main():
  pseudorandom_estimations = []
  pseudorandom_errors = []

  for _ in range(0, 1000):  # 1000 prob
    est, err = pseudorandom(1000)
    pseudorandom_estimations.append(est)
    pseudorandom_errors.append(err)

  quasirandom_estimations = []
  quasirandom_errors = []

  for _ in range(0, 1000):  # 1000 prob
    est, err = quasirandom(1000)
    quasirandom_estimations.append(est)
    quasirandom_errors.append(err)

  plt.figure()
  plt.hist(pseudorandom_estimations, label='pseudorandom')
  plt.hist(quasirandom_estimations, label='quasirandom')
  plt.legend(loc='best')
  plt.title('Estimation differences between different random methods')

  plt.figure()
  plt.hist(pseudorandom_errors, label='pseudorandom')
  plt.hist(quasirandom_errors, label='quasirandom')
  plt.legend(loc='best')
  plt.title('Error differences between different random methods')

  plt.show()


if __name__ == '__main__':
  main()
```

Wyniki
Wyniki najlepiej będzie przedstawić na histogramach





Metoda quasi losowa z zasosowaniem ciągu Sobola jest znacznie lepsza pod względem dokładności liczenia całki. Błędy pomiaru także są nieporównywalnie mniejsze.