# Metody Monte Carlo

Laboratorium 3

## Zadanie 2

### Kod (C++)

```
#include <iostream>
#include <fstream>
#include <random>
#include <math.h>
#include <chrono>
#include <vector>

#define PI 3.141592653589793

std::uniform_real_distribution<float> uniform;
std::random_device rd;
std::mt19937 randomSource(rd());

double standart_twodim_generator(double x, double y)
{
  return (1.0 / (2.0 * PI)) * exp((-1.0) * (pow(x, 2.0) + pow(y, 2.0)) / 2.0);
}

double randomix_2000(double min, double max)
{
  return (double)uniform(randomSource) * max;
}

std::pair<double, double> marsaglii_braya()
{
  double u, v;
  double b = 0;

  do
  {
    u = randomix_2000(0, 1);
    v = randomix_2000(0, 1);
    b = u * u + v * v;
  } while (b >= 1);

  double z = sqrt(-2.0 * log(b) / b);

  return std::pair<double, double>(u * z, v * z);
}

std::pair<double, double> box_mueller()
{
  double u1 = randomix_2000(0, 1);
  double u2 = randomix_2000(0, 1);

  double x = (sqrt((-2) * log(u1)) * cos(2 * PI * u2));
  double y = (sqrt((-2) * log(u1)) * sin(2 * PI * u2));

  return std::pair<double, double>(x, y);
}

void save_to_file(std::string filename, std::vector<std::pair<double, double>>
input)
```

```
{
  std::ofstream ofs;
  ofs.open(filename);

  for (int i = 0; i < input.size(); i++)
  {
    ofs << input[i].first << " " << input[i].second << std::endl;
  }

  ofs.close();
}

int main()
{
  using std::chrono::duration;
  using std::chrono::high_resolution_clock;

  int iterations = 100000;

  std::vector<std::pair<double, double>> box_mueller_values;
  std::vector<std::pair<double, double>> marsaglii_bray_values;

  auto t1 = high_resolution_clock::now();
  for (int i = 0; i < iterations; i++)
  {
    std::pair<double, double> experiment_value = box_mueller();
    box_mueller_values.push_back(experiment_value);
  }
  auto t2 = high_resolution_clock::now();

  /* Getting number of milliseconds as a double. */
  duration<double, std::milli> ms_double = t2 - t1;
  std::cout << "Box mueller experiment took " << ms_double.count() << " ms." <<
std::endl;

  auto t3 = high_resolution_clock::now();
  for (int i = 0; i < iterations; i++)
  {
    std::pair<double, double> experiment_value = marsaglii_braya();
    marsaglii_bray_values.push_back(experiment_value);
  }
  auto t4 = high_resolution_clock::now();

  /* Getting number of milliseconds as a double. */
  duration<double, std::milli> ms_double_2 = t4 - t3;
  std::cout << "Marsaglii Bray experiment took " << ms_double_2.count() << "
ms." << std::endl;

  /* Save to file */
  save_to_file("zad2_box_mueller.txt", box_mueller_values);
  save_to_file("zad2_marsaglii_bray.txt", marsaglii_bray_values);

  return 0;
}
```

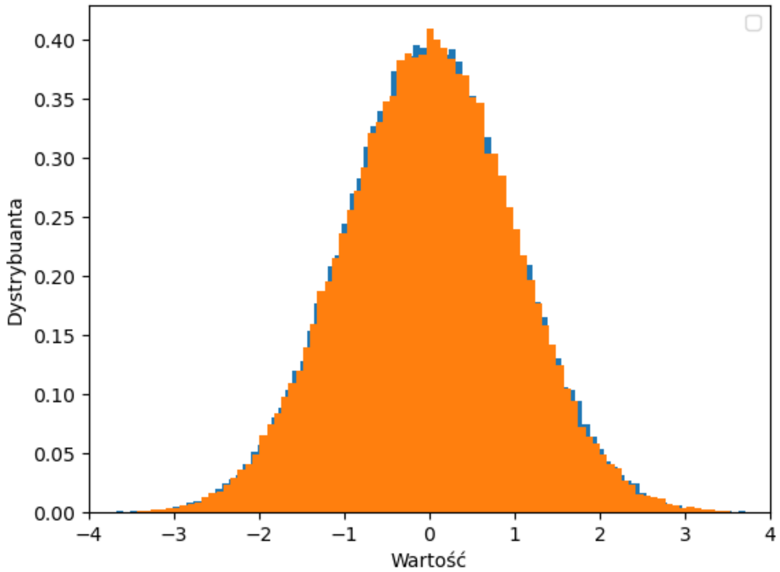
## Kod do tworzenia wykresów (Python)

```python
#!/usr/bin/python

from matplotlib import pyplot as plt
import numpy as np


def print(filename):
  x, y = np.loadtxt(filename, unpack=True)
  plt.xlim([-4, 4])

  plt.hist(x, bins=100, density=True)
  plt.hist(y, bins=100, density=True)

  plt.xlabel('Wartość')
  plt.ylabel('Dystrybuanta')
  plt.legend(loc='best')
  plt.show()


def main():
  print('lab3/zad2_box_mueller.txt')
  print('lab3/zad2_marsaglii_bray.txt')


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

# Wyniki

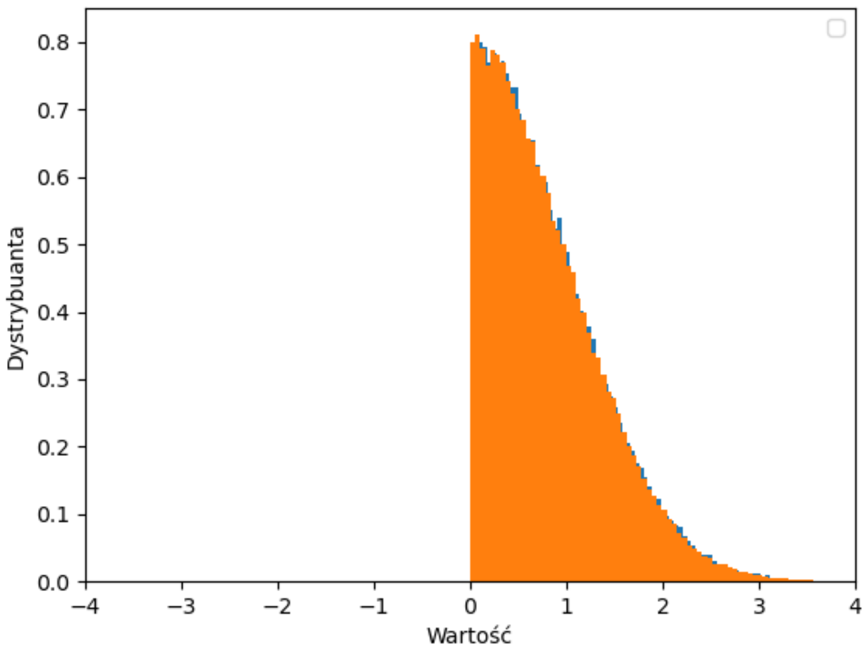
## Box Muller

Box mueller experiment took 25.3592 ms.




## Marsaglii Bray

Marsaglii Bray experiment took 24.7641 ms.




Zgodnie z oczekiwaniem obydwie metody pozwalają stworzyć histogram, przypominający krzywą Gaussa. Obydwie metody wykazują podobną złożoność obliczeniową wykazaną przez pomiary czasu.

# Zadanie 3
## Kod (C++)

```
#include <iostream>
#include <fstream>
#include <random>
#include <math.h>
#include <chrono>
#include <vector>

std::uniform_real_distribution<float> uniform;
std::random_device rd;
std::mt19937 randomSource(rd());

double randomix_2000(double min, double max)
{
  return (double)uniform(randomSource) * max;
}

double random_number_from_probability_density(double x)
{
  return (5.0 / 12.0) * (1.0 + pow(x - 1.0, 4.0));
}

double acceptance_rejection()
{
  double u1, u2;

  do
  {
    u1 = randomix_2000(0, 2);
    u2 = randomix_2000(0, 2);
  } while (u2 >= random_number_from_probability_density(u1));

  return u1;
}

double superposition()
{
  double u1 = randomix_2000(0, 2);
  double u2 = randomix_2000(0, 1);

  if (u2 >= 5.0 / 6.0)
  {
    double sign;
    if (u1 - 1.0 < 0)
    {
      sign = -1.0;
    }
    else
    {
      sign = 1.0;
    }
    return (1.0 + sign * pow(abs(u1 - 1.0), (1.0 / 5.0)));
  }
  else
  {
    return u1;
  }
}

void save_to_file(std::string filename, std::vector<double> input)
{
  std::ofstream ofs;
```

```cpp
  ofs.open(filename);

  for (int i = 0; i < input.size(); i++)
  {
    ofs << input[i] << std::endl;
  }

  ofs.close();
}

int main()
{
  using std::chrono::duration;
  using std::chrono::high_resolution_clock;

  int iterations = 100000;

  std::vector<double> acceptance_rejection_experiment;
  std::vector<double> superposition_experiment;

  auto t1 = high_resolution_clock::now();

  for (int i = 0; i < iterations; i++)
  {
    double experiment_value = acceptance_rejection();
    acceptance_rejection_experiment.push_back(experiment_value);
  }

  auto t2 = high_resolution_clock::now();

  /* Getting number of milliseconds as a double. */
  duration<double, std::milli> ms_double = t2 - t1;
  std::cout << "Acceptance-rejection experiment took " << ms_double.count() << "
ms." << std::endl;

  auto t3 = high_resolution_clock::now();

  for (int i = 0; i < iterations; i++)
  {
    double experiment_value = superposition();
    superposition_experiment.push_back(experiment_value);
  }

  auto t4 = high_resolution_clock::now();

  /* Getting number of milliseconds as a double. */
  duration<double, std::milli> ms_double_2 = t4 - t3;
  std::cout << "Superposition experiment took " << ms_double_2.count() << " ms."
<< std::endl;

  /* Save to file */

  save_to_file("zad3_elimination.txt", acceptance_rejection_experiment);
  save_to_file("zad3_superposition.txt", superposition_experiment);

  return 0;
}
```

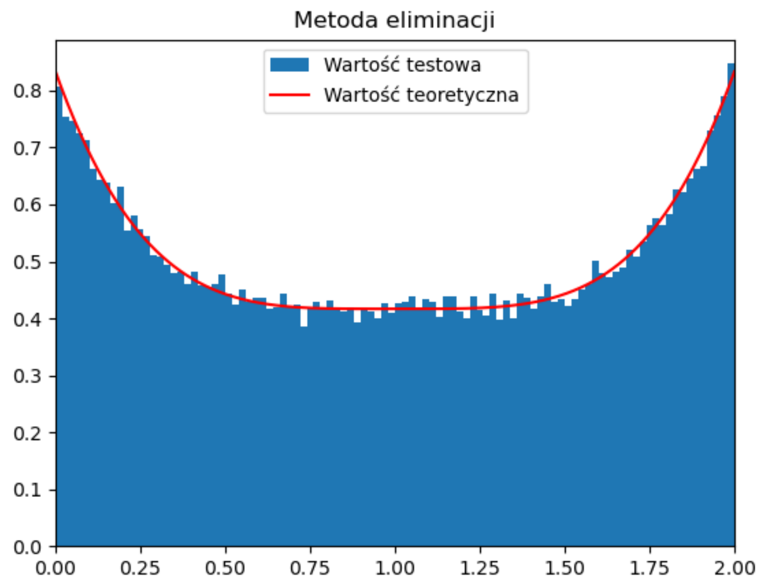
## Kod do tworzenia wykresu (Python)

```python
#!/usr/bin/python

from matplotlib import pyplot as plt
import numpy as np


def teoretical_function(x):
  return 5 / 12 * (1 + (x - 1)**4)


def plot(filename):
  experiment = np.loadtxt(filename, unpack=True)
  x = np.linspace(0, 2, num=100000)
  teoretical = [teoretical_function(i) for i in x]

  plt.figure()
  plt.xlim([0, 2])
  plt.hist(experiment, bins=100, density=True, label='Wartość testowa')
  plt.plot(x, teoretical, 'r-', label='Wartość teoretyczna')
  plt.legend()


def main():
  plot('lab3/zad3_elimination.txt')
  plt.title('Metoda eliminacji')
  plot('lab3/zad3_superposition.txt')
  plt.title('Metoda superpozycji')
  plt.show()


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

## Wyniki

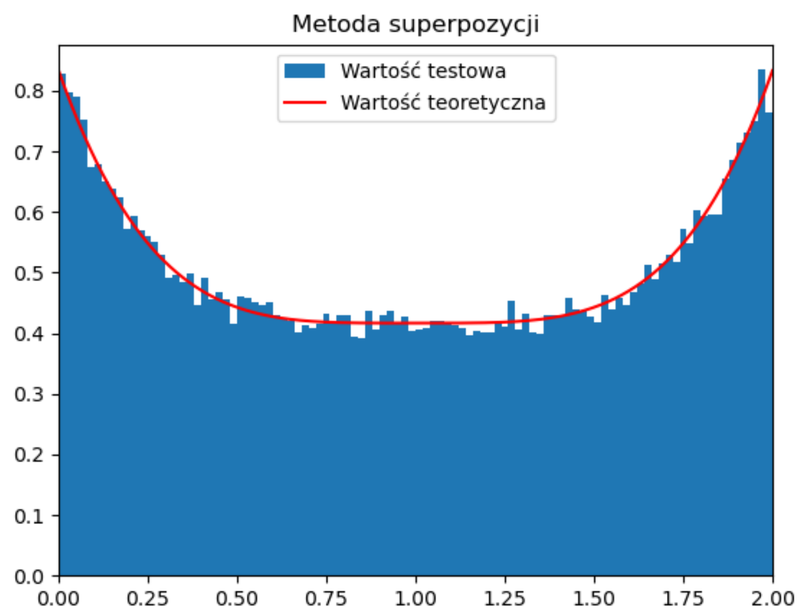
## Metoda eliminacji

Acceptance-rejection experiment took 61.2722 ms.




## Metoda superpozyji

Superposition experiment took 16.1233 ms.




Po zoptymalizowaniu algorytmu wyniki czasowe sugerują, że metoda superpozycji jest około czterokrotnie szybsza od metody eliminacji, przy zachowaniu (optycznej) podobnej dokładności obliczeniowej.
Dla próby 100_000 wartości metoda eliminacji zajmuje około 60 milisekund podczas gdy metoda superpozycji zajmuje średnio 16 milisekund.