## 1 Kod Źródłowy

Implementacja symulacji wrzucania kul do urn oraz badania i zapisywania konkretnych statystyk w języku C++:

```
#include <iostream>
#include <random>
#include <fstream>

// defining simulation parameters
#define MIN_BINS_NUMBER 1000
#define MAX_BINS_NUMBER 100000
#define STEP            1000
#define REPEATS         50


// creating random numbers generator
std::random_device randomDevice();
std::mt19937 randomIntegerGenerator(randomDevice());

// declaring and opening files for saving simulation results
std::ofstream b_n_file("results/bn.csv");
std::ofstream u_n_file("results/un.csv");
std::ofstream l_n_file("results/ln.csv");
std::ofstream c_n_file("results/cn.csv");
std::ofstream d_n_file("results/dn.csv");
std::ofstream dn_minus_cn_file("results/dnminuscn.csv");


// function for one simulation for specific bins numbers
void simulate_for_n_bins(unsigned int bins_number) {
    // setting range of generating numbers
    std::uniform_int_distribution<> distribution(0, bins_number - 1);

    // initilizing empty bins
    unsigned int balls_in_bin[bins_number];
    for(unsigned int i = 0; i < bins_number; i++) {
        balls_in_bin[i] = 0;
    }

    // defining variables
    unsigned int emptyBins = bins_number;
    unsigned int oneBallBins = 0;
    unsigned int twoOrMoreBallBins = 0;
    unsigned int birthdayParadox = 0;
    unsigned int throwsToAllNoEmptyBins = 0;
```

```
// simulating first n(bins_number) throws
for(unsigned int i = 0; i < bins_number; i++) {
    unsigned int bin_numer = distribution(randomIntegerGenerator);

    if(balls_in_bin[bin_numer] == 0) {
        emptyBins--;
        oneBallBins++;
    }
    else if(balls_in_bin[bin_numer] == 1) {
        oneBallBins--;
        twoOrMoreBallBins++;

        if(birthdayParadox == 0) {
            birthdayParadox = i+1;
        }
    }

    balls_in_bin[bin_numer]++;
}

if(emptyBins == 0) {
    throwsToAllNoEmptyBins = bins_number;
}
u_n_file << emptyBins << ',';

unsigned int maxBallNumber = 0;
for(unsigned int i = 0; i < bins_number; i++) {
    if(balls_in_bin[i] > maxBallNumber) {
        maxBallNumber = balls_in_bin[i];
    }
}
l_n_file << maxBallNumber << ',';

// iterating while in every bins are 2 or more balls
unsigned int counter = bins_number;
while(twoOrMoreBallBins != bins_number) {
    counter++;
    unsigned int bin_numer = distribution(randomIntegerGenerator);

    if(balls_in_bin[bin_numer] == 0) {
        emptyBins--;
        oneBallBins++;

        if(emptyBins == 0 && throwsToAllNoEmptyBins == 0) {
            throwsToAllNoEmptyBins = counter;
        }
    }
```

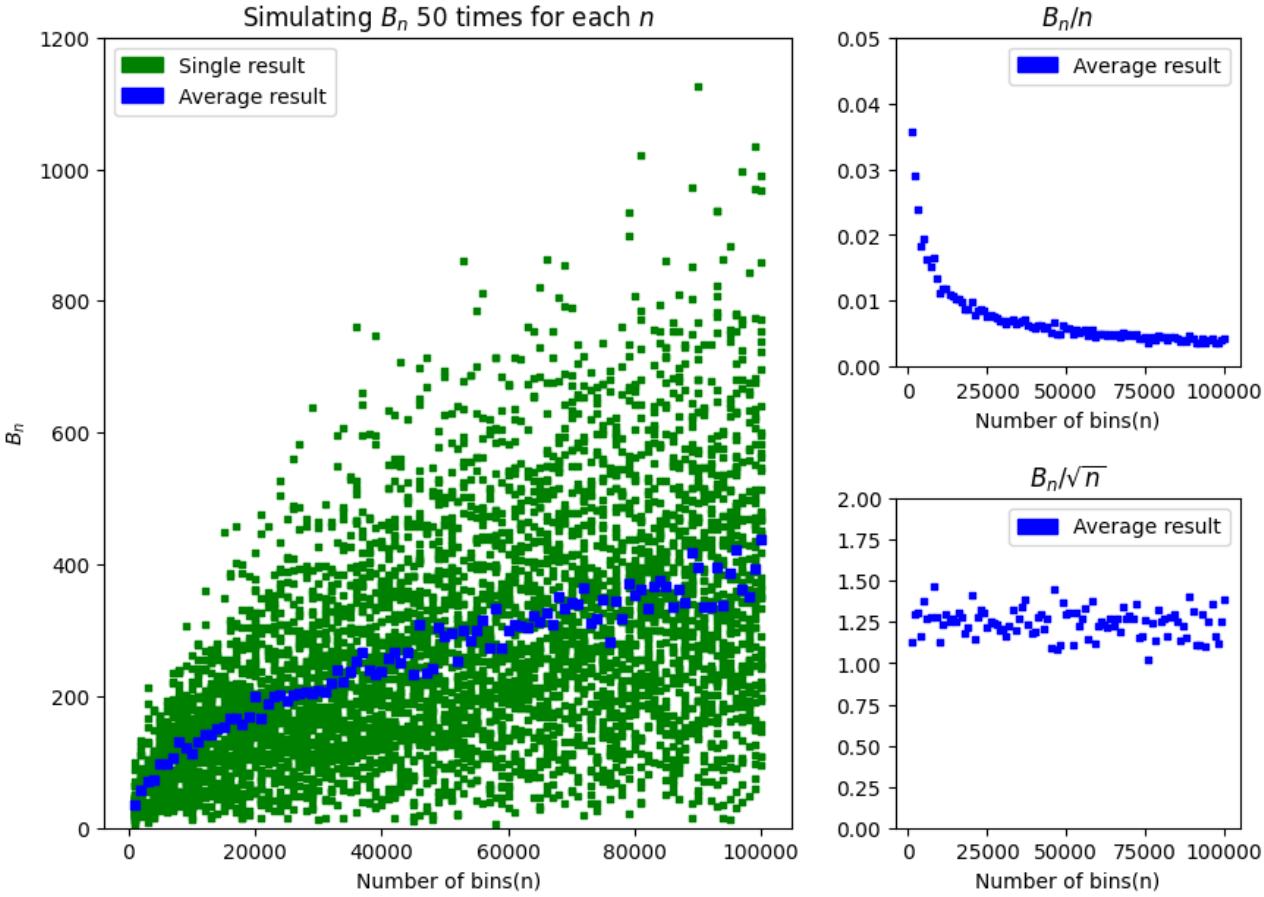
```
        else if(balls_in_bin[bin_numer] == 1) {
            oneBallBins--;
            twoOrMoreBallBins++;

            if(birthdayParadox == 0) {
                birthdayParadox = counter;
            }
        }

        balls_in_bin[bin_numer]++;
    }

    b_n_file << birthdayParadox << ',';
    c_n_file << throwsToAllNoEmptyBins << ',';
    d_n_file << counter << ',';
    dn_minus_cn_file << counter - throwsToAllNoEmptyBins << ',';
}


int main() {
    // main simulation loop
    for(unsigned int i = MIN_BINS_NUMBER; i <= MAX_BINS_NUMBER; i += STEP) {
        for(unsigned int j = 0; j < REPEATS; j++) {
            simulate_for_n_bins(i);
        }

        b_n_file << '\n';
        u_n_file << '\n';
        l_n_file << '\n';
        c_n_file << '\n';
        d_n_file << '\n';
        dn_minus_cn_file << '\n';
    }

    // closing files with simulation results
    b_n_file.close();
    u_n_file.close();
    l_n_file.close();
    c_n_file.close();
    d_n_file.close();
    dn_minus_cn_file.close();

    return 0;
}
```

Implementacja generowania wykresów na podstawie wyznaczonych wcześniej danych w języku Python:

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle


def generate_chart(path_to_data, functions_to_compare, titles, yranges):
    df = pd.read_csv(path_to_data)
    vals = np.array(df.values)
    vals = np.delete(vals, -1, axis=1)
    xs = np.arange(1000, 101000, 1000)

    fig = plt.figure()
    gs = fig.add_gridspec(len(functions_to_compare), 2, width_ratios=[2,1])
    fig.add_subplot(gs[:, 0])

    for i in range(df.shape[1]-1):
        ys = vals.T[i, :]
        plt.scatter(xs, ys, color='g', marker='s', s=10)

    avgs = [sum(row) / len(row) for row in vals]

    plt.scatter(xs, avgs, color='b', marker='s', s=20)
    plt.title('Simulating {} $50$ times for each $n$'.format(titles[0]))
    plt.xlabel('Number of bins(n)')
    plt.ylabel(titles[0])
    plt.ylim(*yranges[0])

    greenExtra = Rectangle((0, 0), 1, 1, color='green')
    blueExtra = Rectangle((0, 0), 1, 1, color='blue')
    plt.legend([greenExtra, blueExtra], ['Single result', 'Average result'])

    for index, fun in enumerate(functions_to_compare):
        fig.add_subplot(gs[index, 1])
        ys = [avg / fun(n) for avg,n in zip(avgs, xs)]
        plt.scatter(xs, ys, color='b', marker='s', s=10)
        plt.title(titles[index + 1])
        plt.ylim(*yranges[index + 1])
        plt.xlabel('Number of bins(n)')
        plt.legend([blueExtra], ['Average result'])

    plt.subplots_adjust(hspace=0.7)
    plt.rcParams["figure.figsize"] = (10, 7)
    plt.show()
```

## 2 Wyniki Eksperymentów

Paradoks urodzinowy, czyli
numer rzutu, w którym kula po raz pierwszy trafia do niepustej urny($B_n$)

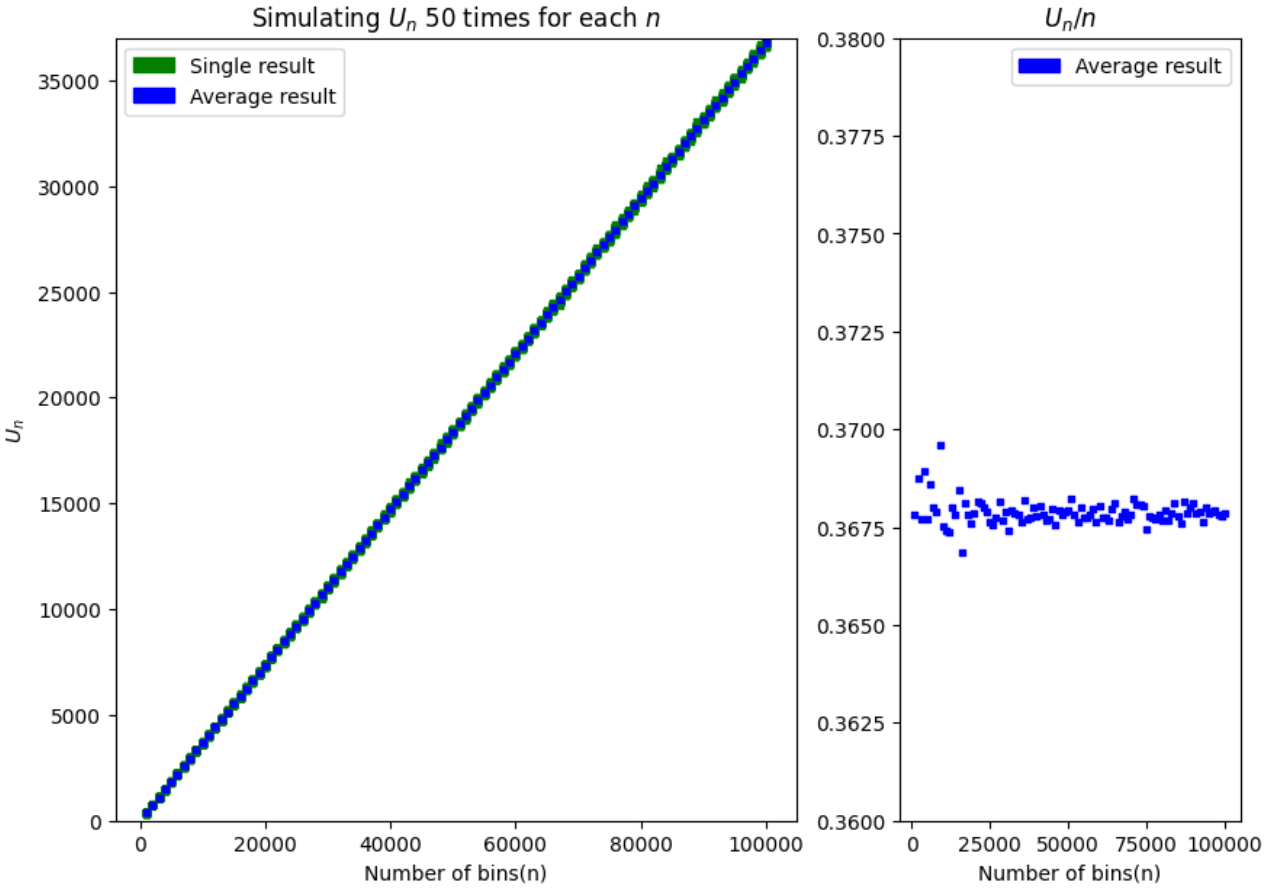



Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: numery rzutów, w których następuje pierwsze wrzucenie kuli do niepustej urny dla każdego z 50 powtórzeń symulacji oraz ich średnie wyznaczone dla każdej liczby urn$(1000 - 100000)$. Gdy liczba urn jest niewielka to koncentracja wyników wokół średniej jest duża. Wraz ze wstrostem liczby urn koncentracja szybko maleje. Dodatkowo wraz ze wzrostem liczby urn przedział w jakim zawarte jest kilka kolejnych średnich zwiększa swoją długość. Wraz ze wzrostem liczby urn rośnie numer kuli, przy której następuje pierwsza kolizja.

Po prawej stronie znajdują się 2 wykresy, które zestawiają ze sobą średnią wartość statystyki $B_n$ oraz funkcje $f(n) = n$ i $f(n) = \sqrt{n}$. Górny wykres pokazuje, że iloraz $\frac{B_n}{n}$ szybko zbiega do 0. Zaś iloraz $\frac{B_n}{\sqrt{n}}$ utrzymuje stałą wartość w zakresie $(1.0, 1.5)$. Na tej podstawie można postawić hipotezę, że:

$$B_n = O(\sqrt{n}).$$

Liczba pustych urn po wrzuceniu $n$ kul($U_n$)

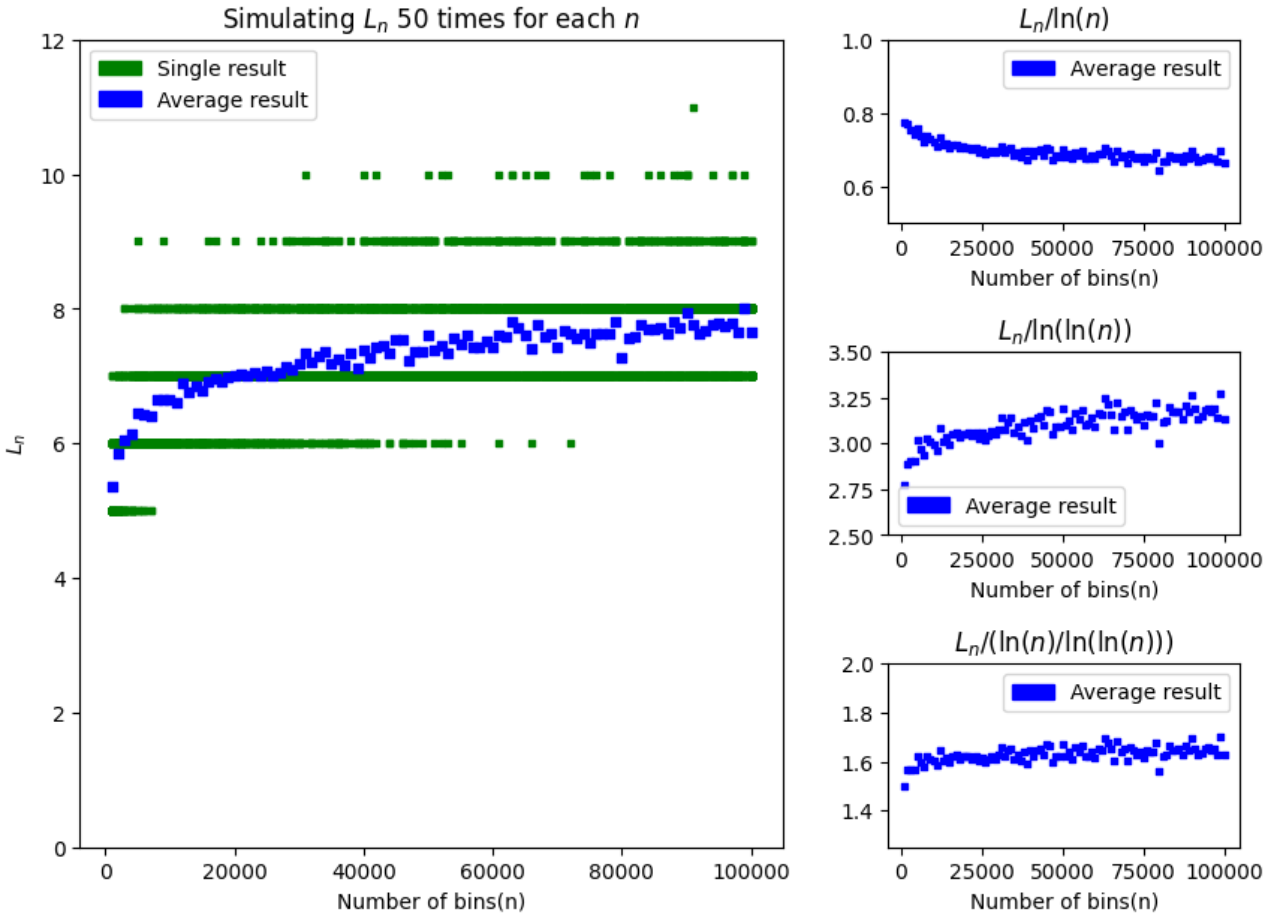



Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: liczba pustych urn po wrzuceniu tylu kul ile jest urn oraz ich średnie wyznaczone dla każdej liczby urn($1000 - 100000$). Koncentracja poszczególnych wyników symulacji wokół wyznaczonej średniej jest bardzo silna. Wraz ze wzrostem liczby urn rośnie liczby pustych urn.

Po prawej stronie znajduje się wykres, który zestawia ze sobą średnią wartość statystyki $U_n$ oraz funkcje $f(n) = n$. Wykres pokazuje, że iloraz $\frac{U_n}{n}$ utrzymuje stałą wartość w okolicach 0.36. Na tej podstawie można postawić hipotezę, że:

$$U_n = O(n).$$

Maksymalne obciążenie, czyli
maksymalna liczba kul w urnie po wrzuceniu $n$ kul($L_n$)

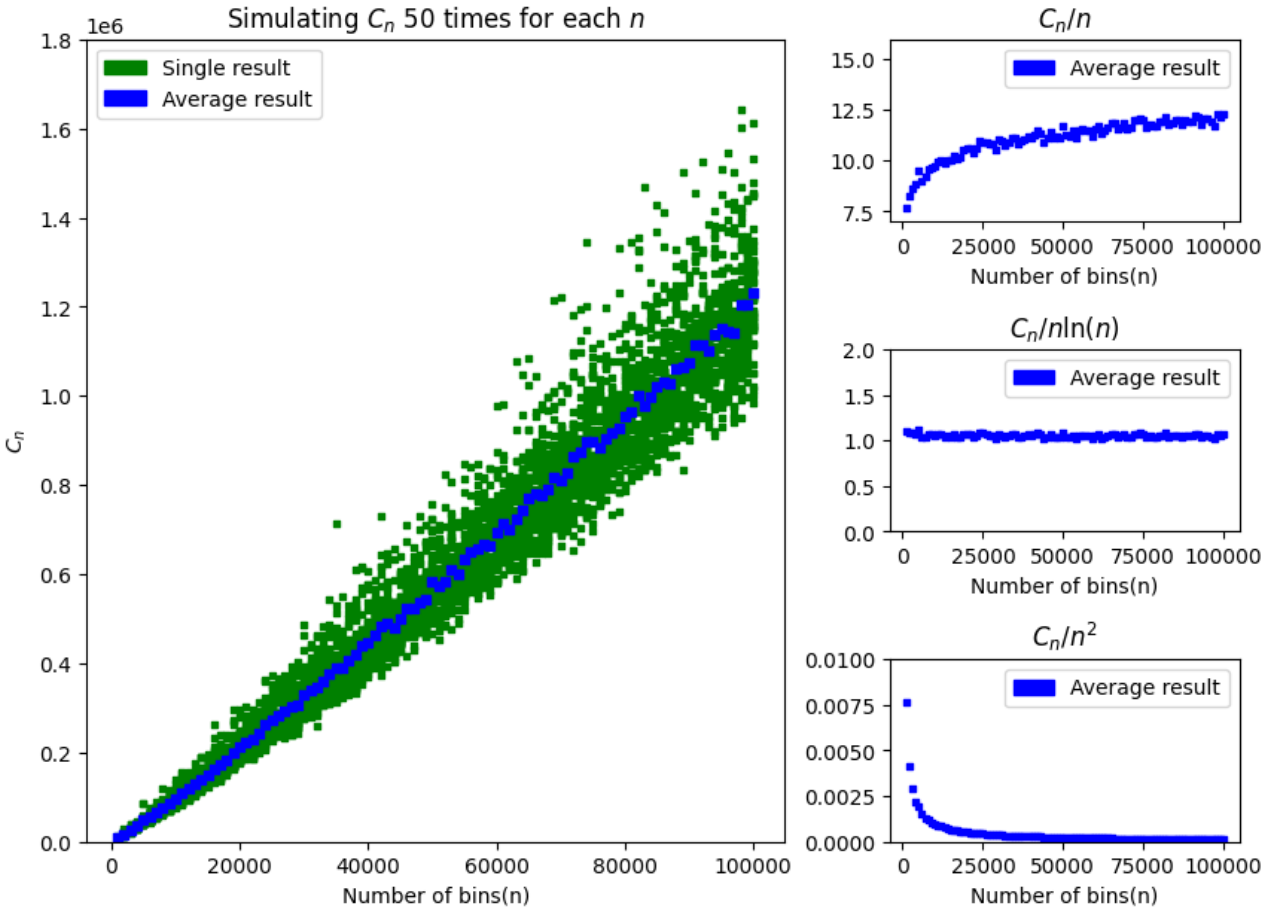



Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: maksymalna liczba kul w urnach po wrzuceniu tylu kul ile jest urn oraz ich średnie wyznaczone dla każdej liczby urn$(1000 - 100000)$. Koncentracja poszczególnych wyników symulacji wokół wyznaczonej średniej jest duża. Wraz ze wzrostem liczby urn dość nieznacznie w porównaniu do liczby urn rośnie maksymalna liczba kul. Liczba wartości, które są przyjmowane jest niewielka.

Po prawej stronie znajdują się 3 wykresy, które zestawiają ze sobą średnią wartość statystyki $L_n$ oraz funkcje $f(n) = \ln(n)$, $f(n) = \ln(\ln(n))$ i $f(n) = \frac{\ln(n)}{\ln(\ln(n))}$. Górny wykres pokazuje, że iloraz $\frac{L_n}{\ln(n)}$ maleje, gdy $n$ rośnie. Środkowy, że iloraz $\frac{L_n}{\ln(\ln(n))}$ rośnie, gdy $n$ rośnie. Zaś dolny, że iloraz $\frac{L_n}{\ln(n)/\ln(\ln(n))}$ utrzymuje stałą wartość. Na tej podstawie można postawić hipotezę, że:

$$L_n = O(\tfrac{\ln(n)}{\ln(\ln(n))}).$$

Problem kolekcjonera kuponów, czyli
minimalna liczba rzutów, po której w każdej urnie bedzie conajmniej jedna kula($C_n$)

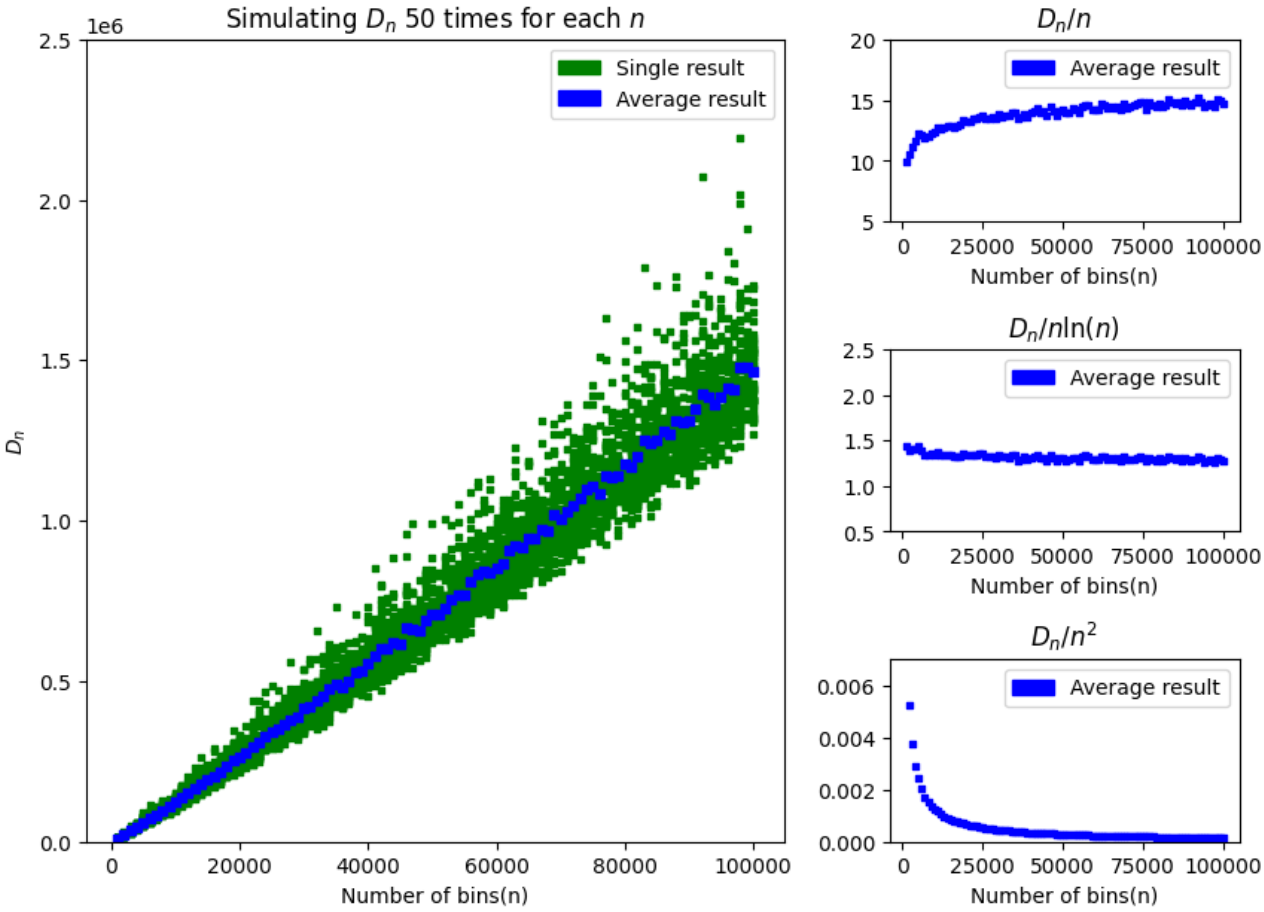



Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: liczba wrzuconych kul, po której w każdej urnie jest co najmniej 1 kula oraz ich średnie wyznaczone dla każdej liczby urn$(1000 - 100000)$. Gdy liczba urn jest niewielka to koncentracja wyników wokół średniej jest duża. Wraz ze wstrostem liczby urn koncentracja maleje. Wraz ze wzrostem liczby urn rośnie liczba potrzebnych kul, aby w każdej z urn znajdowała się conajmniej 1 kula.

Po prawej stronie znajdują się 3 wykresy, które zestawiają ze sobą średnią wartość statystyki $C_n$ oraz funkcje $f(n) = n$, $f(n) = n\ln(n)$ i $f(n) = n^2$. Górny wykres pokazuje, że iloraz $\frac{C_n}{n}$ rośnie, gdy $n$ rośnie. Dolny, że iloraz $\frac{C_n}{n^2}$ szybko zbiega do 0, gdy $n$ rośnie. Zaś środkowy, że iloraz $\frac{C_n}{n\ln(n)}$ utrzymuje stałą wartość w okolicach wartości 1. Na tej podstawie można postawić hipotezę, że:

$$C_n = O(n\ln(n)).$$

Problem brata kolekcjonera kuponów, czyli
minimalna liczba rzutów, po której w każdej urnie będą conajmniej dwie kule($D_n$)

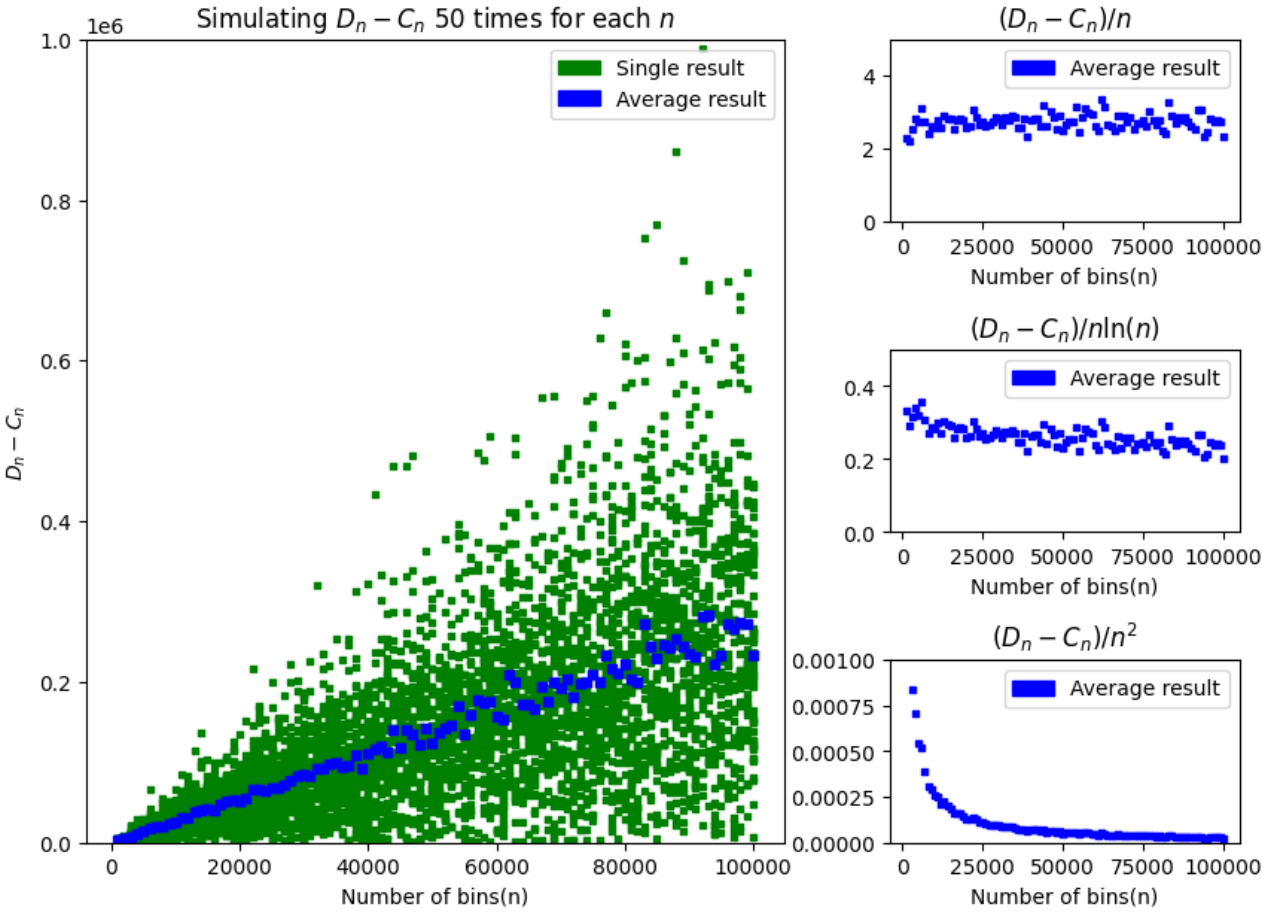



Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: liczba wrzuconych kul, po której w każdej urnie są co najmniej 2 kule oraz ich średnie wyznaczone dla każdej liczby urn$(1000 - 100000)$. Gdy liczba urn jest niewielka to koncentracja wyników wokół średniej jest duża. Wraz ze wstrostem liczby urn koncentracja maleje. Wraz ze wzrostem liczby urn rośnie liczba potrzebnych kul, aby w każdej z urn znajdowały się conajmniej 2 kule.

Po prawej stronie znajdują się 3 wykresy, które zestawiają ze sobą średnią wartość statystyki $D_n$ oraz funkcje $f(n) = n$, $f(n) = n\ln(n)$ i $f(n) = n^2$. Górny wykres pokazuje, że iloraz $\frac{D_n}{n}$ rośnie, gdy $n$ rośnie. Dolny, że iloraz $\frac{D_n}{n^2}$ szybko zbiega do 0, gdy $n$ rośnie. Zaś środkowy, że iloraz $\frac{D_n}{n\ln(n)}$ utrzymuje stałą wartość w okolicach wartości 1.5. Na tej podstawie można postawić hipotezę, że:

$$D_n = O(n\ln(n)).$$

Lczba rzutów potrzebna, aby przejść z $C_n$ do $D_n$




Powyższe wykresy przedstawiają wyniki jednej ze statystyk z przeprowadzonych symulacji. Po lewej stronie: liczba kul potrzebna, aby przejść z sytuacji opisanej dla statystyki $C_n$ do sytuacji opisanej w statystyce $D_n$ oraz ich średnie wyznaczone dla każdej liczby urn$(1000 - 100000)$. Gdy liczba urn jest niewielka to koncentracja wyników wokół średniej jest duża. Wraz ze wstrostem liczby urn koncentracja szybko maleje. Wraz ze wzrostem liczby urn rośnie liczba potrzebnych kul.

Po prawej stronie znajdują się 3 wykresy, które zestawiają ze sobą średnią wartość statystyki $D_n - C_n$ oraz funkcje $f(n) = n$, $f(n) = n\ln(n)$ i $f(n) = n^2$. Dolny wykres pokazuje, że iloraz $\frac{D_n - C_n}{n^2}$ szybko zbiega do 0, gdy $n$ rośnie. Wykresy górny i dolny, dotyczące odpowiednio ilorazu $\frac{D_n - C_n}{n}$ oraz $\frac{D_n - C_n}{n\ln(n)}$ nie pozwalają na jednoznaczne określenie hipotezy. Zatem eksperyment powinien być przeprowadzony na podstawie większej liczby urn. Na podstawie powtórzonego eksperymentu można postawić hipotezę, że:

$$D_n - C_n = O(n\ln(n))$$

## 3 Intuicje

### Problem kolekcjonera kuponów

- Statystykę $C_n$ można zilustrować w następujący sposób: kolekcjoner zbiera kolekcje unikatowych etykiet, która ma $n$ elementów. Może je zdobyć kupując pewien produkt. Nie widzi jednak, jaką etykietę zawiera dany produkt. Ile produktów musi kupić kolekcjoner, aby zdobyć wszystkie etykiety. Odpowiedź brzmi: $C_n$.
- Problem kolekcjonera polega na tym, że im więcej etykiet już posiada, tym trudniej jest zdobyć nową etykietę.

### Paradoks urodzinowy

- Statystykę $B_n$ można zilustrować w następujący sposób: grupa osób spotyka się na przyjęciu. Każda z nich podaje dzień i miesiąc urodzenia. Ile osób musi być, aby prawdopodobieństwo tego zdarzenia było w miarę duże. (liczba urn = 365 dni roku)
- Paradosk polega na tym, że wystarczy 23 osoby, aby mieć 50% szansy, natomiast 50 osób gwarantuje ponad 90% szansy na dwie osoby urodzone w tym samym dniu roku.
- W kontekście funkcji haszujących statystyka $B_n$ określa średnią liczbę kluczy, które będą miały różne wyniki po przejsciu przez funkcje haszująca, gdy mamy $n$ różnych mozliwych wartości tej funkcji. Im większa wartość $B_n$, tym lepsza/bezpieczniejsza funkcja haszująca.