Analiza algorytmów. Lista 3

Piotr Berezowski, 236749

26 kwietnia 2020

1 Implementacja zadań

Załączone pliki:

- *unique_sum.go* - zawiera implementację algorytmu przybliżonego sumowania wykożystywanego w zadaniu 9.
- *unique_avg.go* - zawiera implementację algorytmu liczącego średnią unikalnych elementów (zadanie 10).
- *generators.go* - zawiera funkcje używane do generowania zbiorów.
- *zad1.go, zad2.go* - funkcje generujące dane do wykresów.
- *main.go* - zawiera testowane funkcje haszujące i wszystkie funkcje pomocnicze.
- *plots.py* - zawiera funkcje rysujące wykresy.

2 Zadanie 9

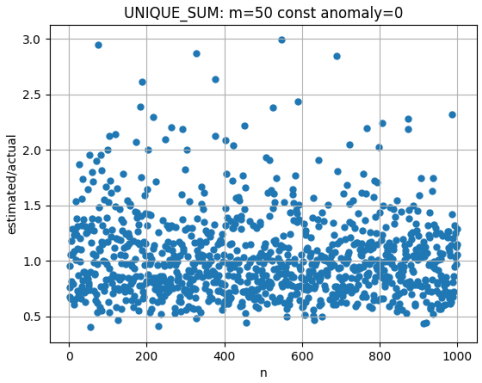
2.1 Opis zadania

Przeczytaj notatki do wykładu i zaimplementuj opisany tam algorytm przybliżonego sumowania. Wykonaj eksperymenty analogiczne do tych dla przybliżonego zliczania. W szczególności:

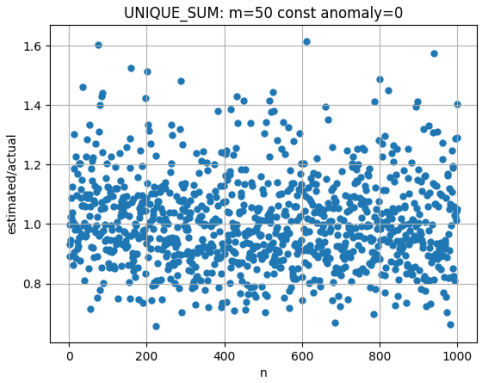
- wykorzystaj metodę odwrotnej dystrybuanty i przetestuj różne funkcje haszujące
- sprawdź dokładność algorytmu dla różnych scenariuszy, na przykład kiedy wartościcech $\lambda_1, \lambda_2, \ldots, \lambda_n$ są takie same, losowo wybrane z rozkładu jednostajnego na przedziale (a, b) dla różnych a i b, większość wartości jest podobna ale istnieją wartości odstające
- porównaj wyniki eksperymentów z ograniczeniami wynikającymi z nierówności Czebyszewa.

2.2 Rozwiązanie

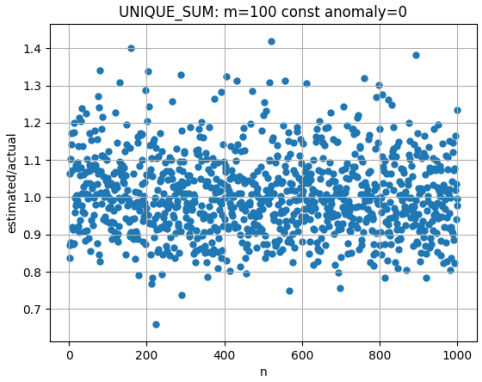
2.2.1 Wyniki dla różnych wartości parametru m



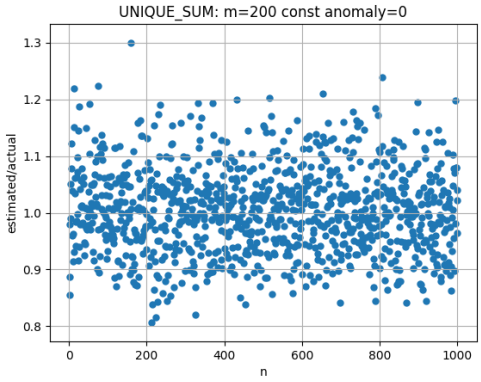
(a) m = 10



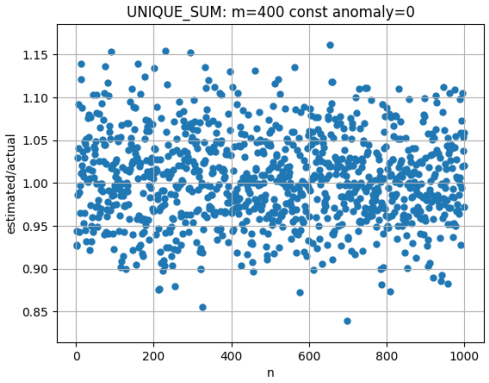
(b) m = 50



(c) m = 100



(d) m = 200

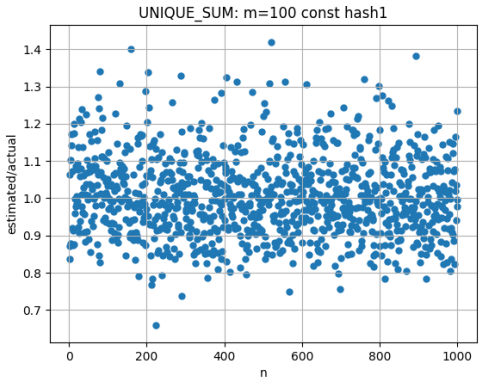


(e) m = 400

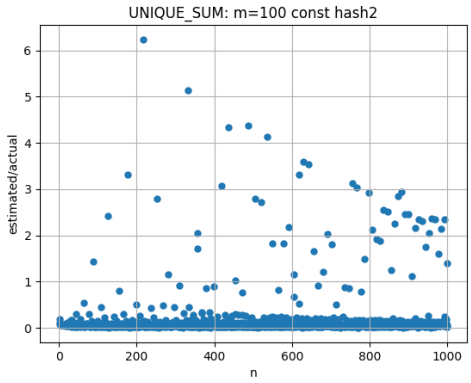
Rysunek 1: Wykresy dla $\lambda = \text{const}$ i funkcji haszującej $hash1$

Widzimy, że wraz ze wzrostem wartości m rośnie dokładność z jaką szacowana jest suma.

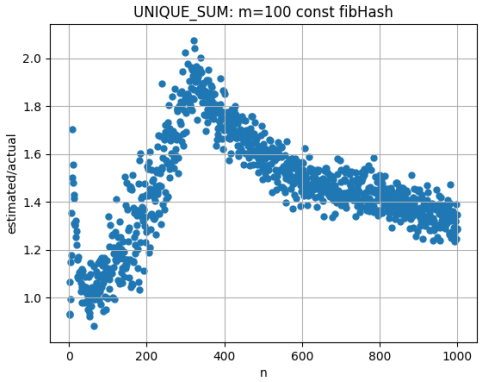
2.2.2 Wyniki dla różnych funkcji haszujących



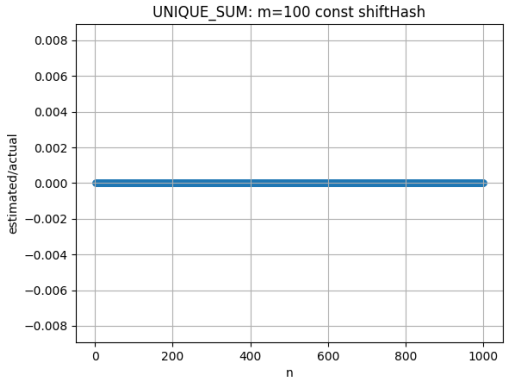
(a) $h = hash1$



(b) $h = hash2$



(c) $h = fibHash$

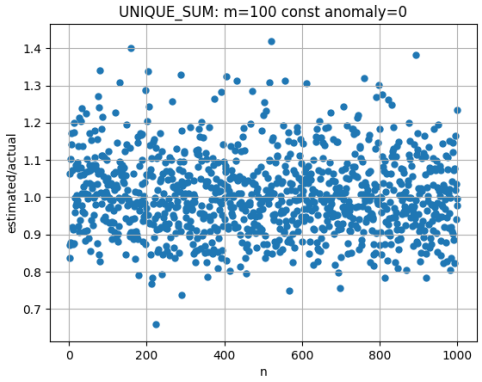


(d) $h = shiftHash$

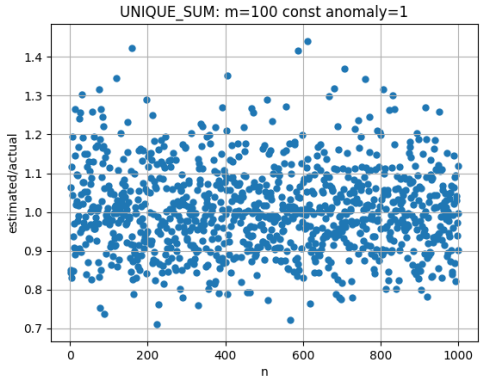
Rysunek 2: Różne funkcje haszujące dla wartości $\lambda = \text{const}$ i $m = 100$

Widzimy, że dla "złych" funkcji haszujących, które są nieodporne na kolizje, lub nierównomiernie "wypełniają" zbiór wartości funkcji estymowana suma była znacznie gorszej jakości niż dla "dobrej" funkcji haszującej jaką jest funkcja etykietowana jako $hash1$.

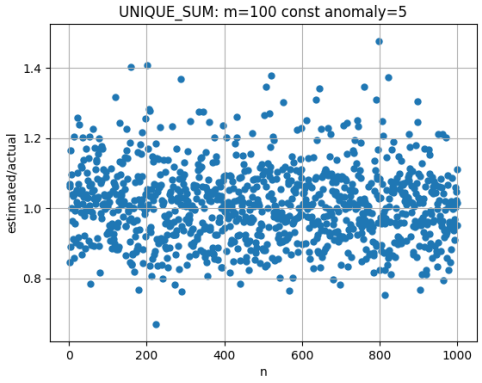
2.2.3 Wyniki dla różnych wartości λ



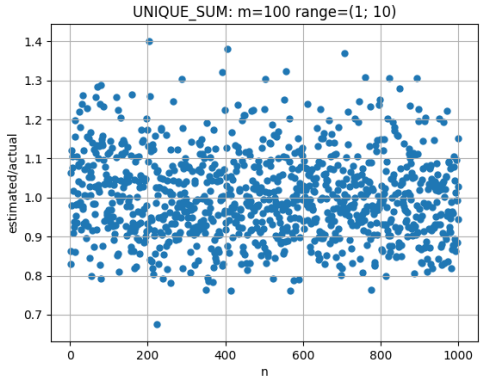
(a) $\lambda = \text{const}$



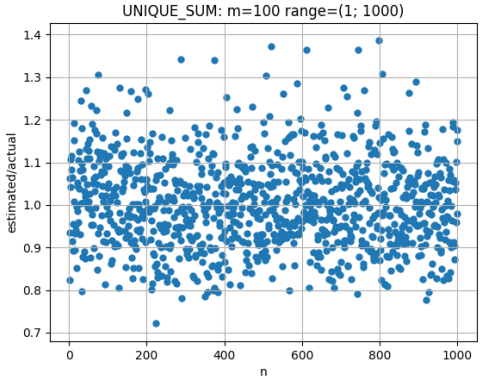
(b) $\lambda = \text{const}$, 1% wartości odstających



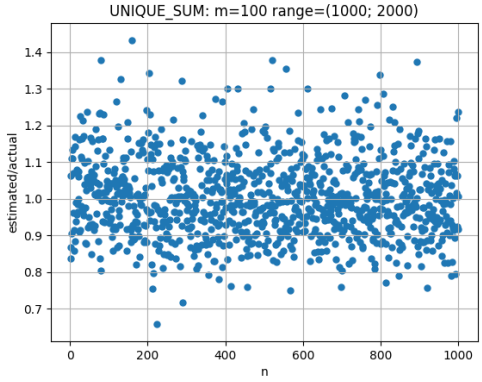
(c) $\lambda = \text{const}$, 5% wartości odstających



(d) $\lambda \in [1, 10]$



(e) $\lambda \in [1, 1000]$



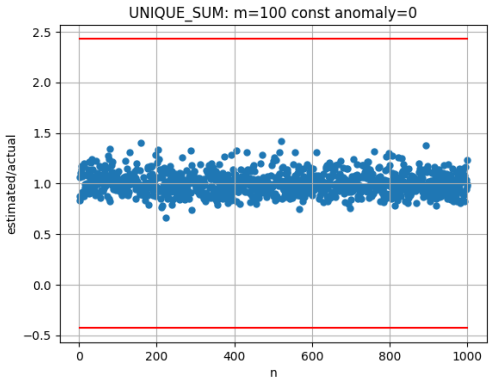
(f) $\lambda \in [1000, 2000]$

Rysunek 3: Wykresy dla $m = 100$ i funkcji haszującej $h = hash1$

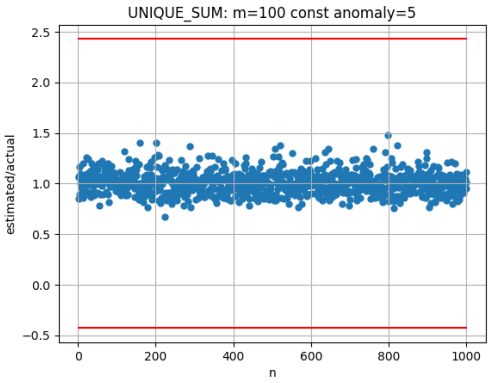
Dokładność estymowanej sumy jest podobna, niezależnie od tego w jaki sposób wybrane zostały wartości λ elementów zbioru.

2.2.4 Nierówność Czebyszewa

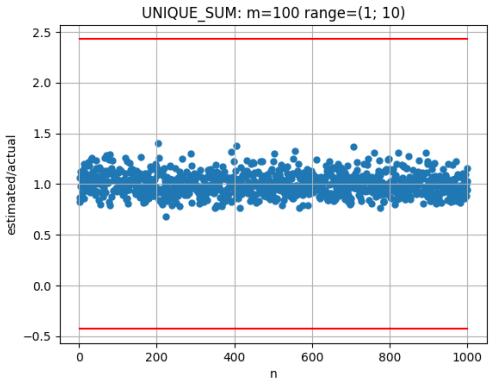
$\alpha = 0.5\%$:



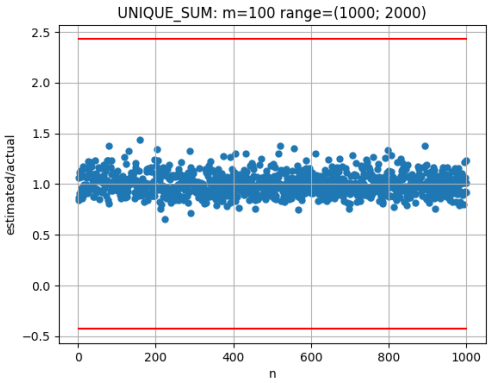
(a) $\lambda = \text{const}$



(b) $\lambda = \text{const}$, 5% wartości odstających



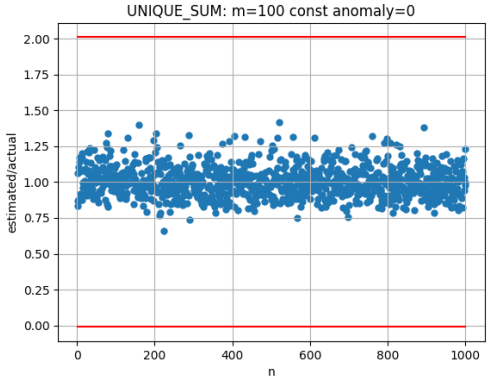
(c) $\lambda \in [1, 10]$



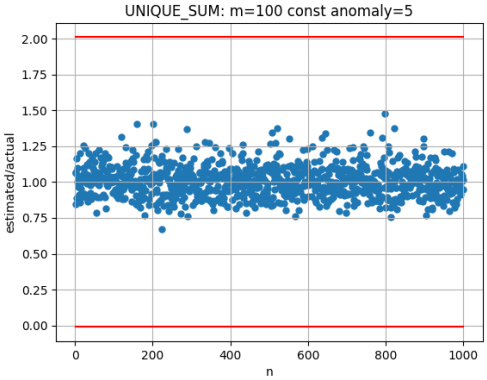
(d) $\lambda \in [1000, 2000]$

Rysunek 4: Wykresy dla $m = 100$ i funkcji haszującej $h = hash1$ (ograniczenie z nierówności Czebyszewa dla parametru $\alpha = 0.5\%$)

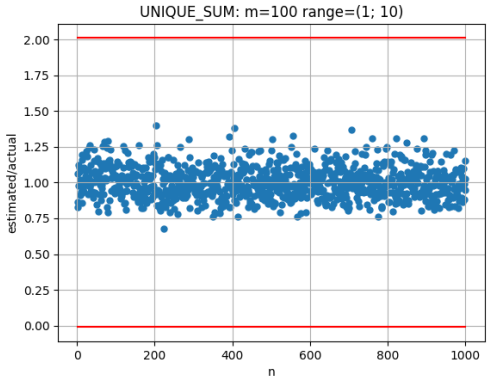
$\alpha = 1\%$:



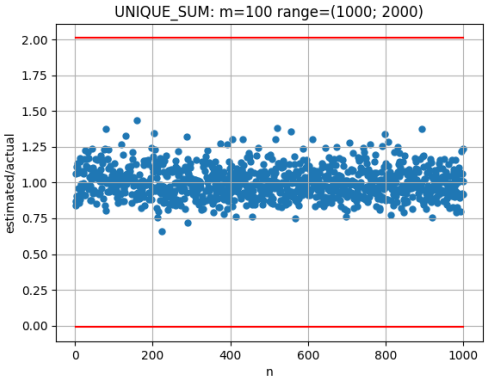
(a) $\lambda = \text{const}$



(b) $\lambda = \text{const}$, 5% wartości odstających



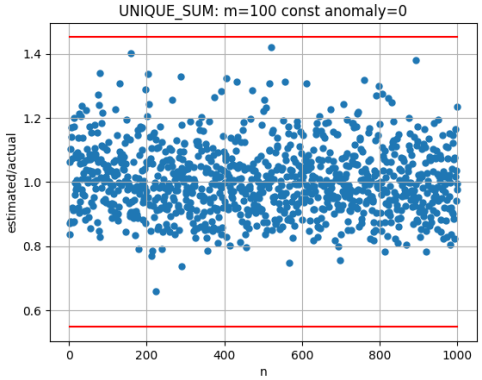
(c) $\lambda \in [1, 10]$



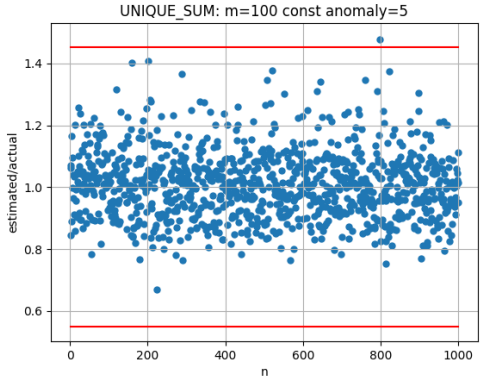
(d) $\lambda \in [1000, 2000]$

Rysunek 5: Wykresy dla $m = 100$ i funkcji haszującej $h = hash1$ (ograniczenie z nierówności Czebyszewa dla parametru $\alpha = 1\%$)

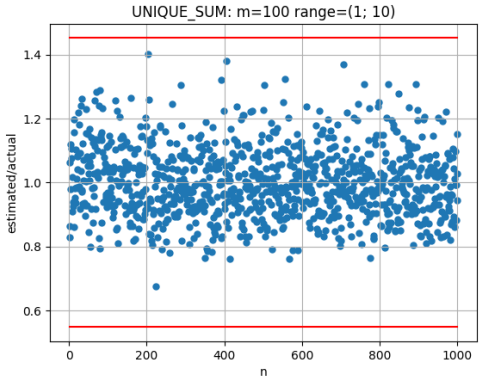
$\alpha = 5\%$:



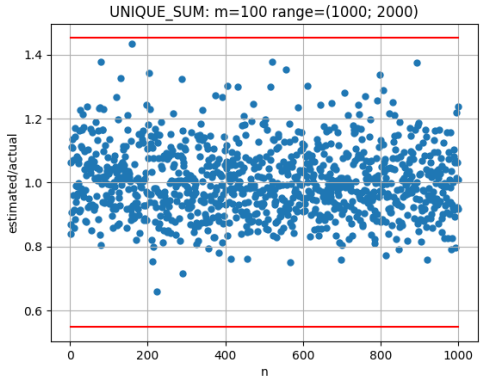
(a) $\lambda = \text{const}$



(b) $\lambda = \text{const}$, 5% wartości odstających



(c) $\lambda \in [1, 10]$



(d) $\lambda \in [1000, 2000]$

Rysunek 6: Wykresy dla $m = 100$ i funkcji haszującej $h = hash1$ (ograniczenie z nierówności Czebyszewa dla parametru $\alpha = 5\%$)

3 Zadanie 10

3.1 Opis zadania

Zaproponuj i przetestuj procedurę, która umożliwi w oszacowanie średniej wartości unikalnych elementów

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_n}{n}$$

w ramach jednego przebiegu po multizbiorze.

3.2 Rozwiązanie

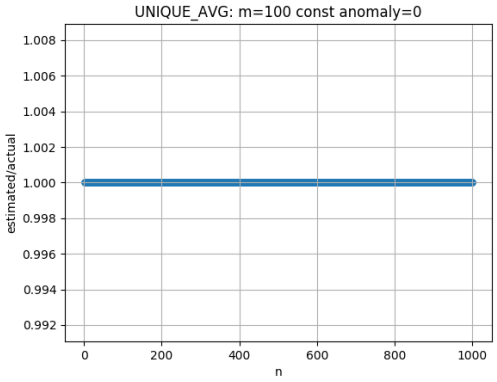
Proponowany algorytm jest modyfikacją algorytmu UNIQUE_SUM. Modyfikacja polega na tym, że oprócz tablicy M wprowadzamy tablicę N, która będzie służyć estymacji liczby wszystkich unikalnych elementów multizbioru. Podobnie jak w przypadku tablicy zawartości tablicy M, zawartość N jest uaktualniana dla każdego elementu multizbioru w sposób przedstawiony na poniższym pseudokodzie:

1: UniqueAvg($\mathfrak{M}, h, m$)

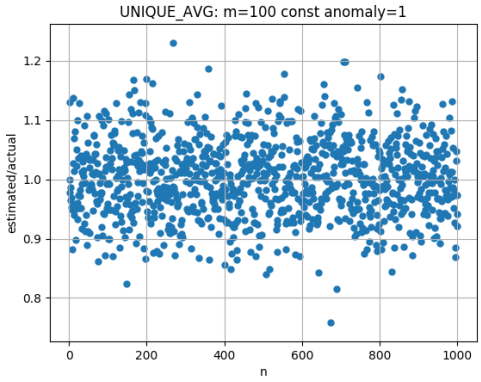
Input : $\mathfrak{M}$, h, m
Output: avg - średnia wartość unikalnych elementów

Set each of m positions of M and N to ∞;
for $(i, \lambda_i) \in \mathfrak{M}$ **do**
    for $k \in \{1, 2, \ldots, m\}$ **do**
        $u \leftarrow h(i^\frown k)$;
        $M[k] \leftarrow \min(M[k], -\frac{\ln u}{\lambda_i})$;
        $N[k] \leftarrow \min(N[k], -\frac{\ln u}{1})$;
$\bar{\Lambda} \leftarrow \frac{m-1}{\sum_{k=1}^{m} M[k]}$;
$\bar{c} \leftarrow \frac{m-1}{\sum_{k=1}^{m} N[k]}$;
return $avg = \frac{\bar{\Lambda}}{\bar{c}}$

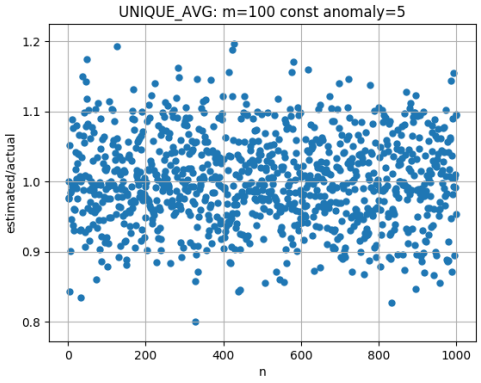
3.2.1 Wyniki dla różnych wartości λ



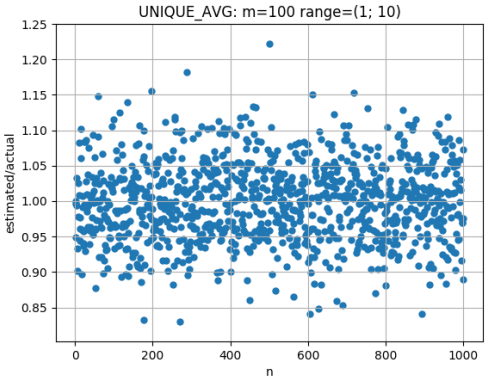
(a) $\lambda = \text{const}$



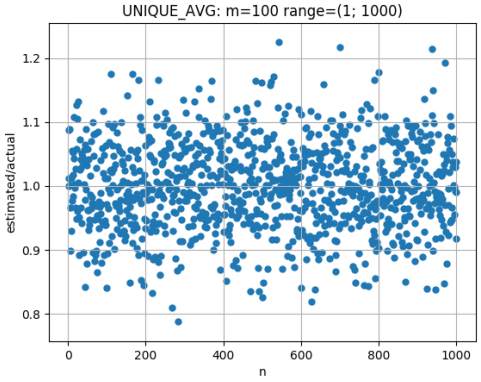
(b) $\lambda = \text{const}$, 1% wartości odstających



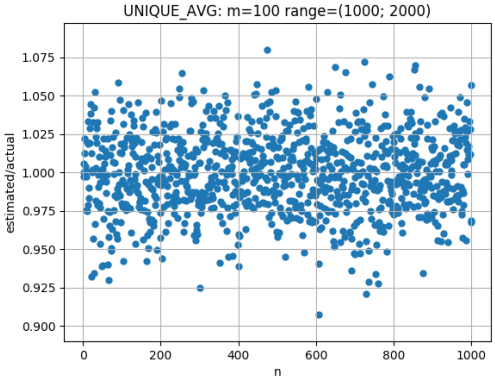
(c) $\lambda = \text{const}$, 5% wartości odstających



(d) $\lambda \in [1, 10]$



(e) $\lambda \in [1, 1000]$



(f) $\lambda \in [1000, 2000]$

Rysunek 7: Wykresy dla $m = 100$ i funkcji haszującej $h = hash1$