Algorytm szybkiego badania własności statystycznych podciągów szeregu liczbowego

Andrzej Chmielowiec

14 maja 2018

Streszczenie

W artykule zaprezentowano algorytm szybkiego badania podstawowych właściwości statystycznych (wartość oczekiwana i wariancja) wszystkich podciągów danego ciągu liczbowego. Dodatkowo pokazano, że jeśli mamy do czynienia z danymi pomiarowymi, to do wykonywania tego rodzaju obliczeń statystycznych można wykorzystać zarówno reprezentację zmiennoprzecinkową, jak i całkowitoliczbową. Ze względu na charakter analizowanych obliczeń przedstawione zostały warunki konieczne, które muszą być spełnione przez typ liczbowy, aby wszystkie badane statystyki wyznaczone zostały praktycznie bez błędu numerycznego. Artykuł prezentuje też czas działania poszczególnych algorytmów ze względu typ liczbowy, który została wykorzystany.

1 Wprowadzenie

Badanie statystycznych własności określonego szeregu danych pomiarowych jest bardzo dobrze zbadanym zagadnieniem. Dysponujemy wieloma sprawdzonymi metodami do weryfikacji określonych hipotez statystycznych. Wszystkie one zakładają jednak, że mamy do dyspozycji dobrze określony ciąg danych, które poddajemy sprawdzeniu. Z zupełnie inną sytuacją mamy do czynienia w przypadku, gdy wewnątrz pewnego ciągu danych poszukujemy podciągów posiadających określone własności statystyczne. Ze względu na liczbę potencjalnych możliwości pojawia się problem dużej złożoności obliczeniowej podejścia, polegającego na niezależnej analizie wszystkich możliwych podciągów. Dla zobrazowania tego faktu posłużymy się przykładem wyznaczenia średniej arytmetycznej dla wszystkich możliwych podciągów ciągu złożonego z 10^5 pomiarów. Można obliczyć, że wykonanie tego zadania pochłonie w przybliżeniu $1.7 \cdot 10^{14}$ operacji zmiennoprzecinkowych. Zakładając, że pojedynczy rdzeń typowego procesora jest w stanie wykonać

obecnie około $5 \cdot 10^9$ operacji, to czas obliczeń wyniósłby około 9.5 godziny. W związku z tym, zasadne jest poszukiwanie takich rozwiązań algorytmicznych, które pozwolą na istotne przyspieszenie tego rodzaju analiz.

2 Algorytm szybkiego badania własności statystycznych podciągów

W tej części będziemy zakładali, że dysponujemy pewnym ciągiem n obserwacji

$$t = (t_1, t_2, \ldots, t_{n-1}, t_n).$$

Przez podciąg s ciągu t będziemy rozumieli ciąg $s = (t_{l+1}, t_{l+2}, \ldots, t_{l+k})$ o tej własności, że dla każdego i spełniającego warunek $l + 1 \leq i \leq l + k$ zachodzi $t_i \in s$. Będziemy też przyjmowali, że kolejność elementów podciągu s jest taka sama, jak w ciągu t. Ponieważ podciąg jest jednoznacznie określony przez indeks swojego pierwszego i ostatniego wyrazu, to będziemy również używali notacji $s_{l+1,l+k}$ na oznaczenie podciągu $(t_{l+1}, t_{l+2}, \ldots, t_{l+k})$.

Niech T_k oznacza liczbę podciągów długości k ciągu s. Można zauważyć, że liczba ta wyraża się następującym wzorem $T_k = n + 1 - k$. W związku z tym liczba wszystkich możliwych podciągów T ciągu t wynosi

$$T = \sum_{k=1}^{n} T_k = \sum_{k=1}^{n} (n + 1 - k) = n(n + 1) - \frac{n(n+1)}{2} = \frac{n(n+1)}{2}.$$

Teraz przejdziemy do obliczenia złożoności algorytmu wyznaczania średniej arytmetycznej i wariancji dla wszystkich możliwych podciągów ciągu n-elementowego. W pierwszej kolejności zauważmy, że jeśli dany jest podciąg k-elementowy $s = (t_{l+1}, t_{l+2}, \ldots, t_{l+k})$, to zarówno wyznaczenie średniej arytmetycznej, jak i wariancji jest proporcjonalne do liczby jego elementów. Wynika to bezpośrednio z definicji tych wartości:

$$\begin{aligned} E(s) &= \frac{1}{k}\left(t_{l+1} + t_{l+2} + \cdots + t_{l+k}\right), \\ D^2(s) &= \frac{1}{k}\left(t_{l+1}^2 + t_{l+2}^2 + \cdots + t_{l+k}^2\right) - E(s)^2. \end{aligned}$$

Do wyznaczenia wartości oczekiwanej potrzeba $k-1$ dodawań i jedno dzielenie (w sumie k operacji), a do wyznaczenia wariancji potrzeba $k-1$ dodawań, $k + 1$ kwadratowań, jedno odejmowanie i jedno dzielenie (w sumie $2k + 2$ operacje). Możemy więc przyjąć, że liczba operacji zmiennoprzecinkowych

potrzebnych do wyliczenia wartości oczekiwanej i wariancji jest ograniczona przez Ck, gdzie C jest pewną stałą, a k jest liczbą elementów podciągu. W związku z powyższym złożoność algorytmu wyznaczającego wartość oczekiwaną i wariancję wszystkich możliwych podciągów ciągu n-elementowego możemy zapisać jako:

$$\begin{aligned} R_n &= \sum_{k=1}^{n} T_k \cdot Ck \\ &= C \sum_{k=1}^{n} (n+1-k)k \\ &= C(n+1) \sum_{k=1}^{n} k - C \sum_{k=1}^{n} k^2 \\ &= C \left(\frac{n(n+1)^2}{2} - \frac{n(n+1)(2n+1)}{6} \right) \\ &= \frac{C}{6} n(n+1)(n+2) = O(n^3). \end{aligned}$$

Zauważmy, że jedynymi operacjami, które w istotny sposób wpływają na złożoność wyznaczania wartości oczekiwanej i wariancji są: wyznaczanie sumy i wyznaczanie sumy kwadratów. Tylko te operacje zależą od liczby elementów podciągu. Liczba pozostałych operacji jest stała i niezależna od tego, jak długi ciąg przetwarzamy. W związku z tym jedynym sposobem na obniżenie złożoności obliczeniowej przytoczonego powyżej algorytmu jest przyspieszenie wyznaczania sum i sum kwadratów. Okazuje się, że jeśli podciągi będą przetwarzane w odpowiedniej kolejności, to wyznaczanie kolejnych wartości oczekiwanych i wariancji może zostać zrealizowane w stałym czasie.

Ustawmy wszystkie podciągi w takiej kolejności jak to pokazano poniżej:

$$\begin{array}{llllll} (t_1), & (t_1,t_2), & \ldots, & (t_1,\ldots,t_{n-2}), & (t_1,\ldots,t_{n-1}), & (t_1,\ldots,t_n), \\ (t_2), & (t_2,t_3), & \ldots, & (t_2,\ldots,t_{n-1}), & (t_2,\ldots,t_n), & \\ (t_3), & (t_3,t_4), & \ldots, & (t_3,\ldots,t_n), & & \\ \vdots & & & & & \\ (t_{n-1}), & (t_{n-1},t_n), & & & & \\ (t_n). & & & & & \end{array}$$

Zauważmy, że każdy rząd rozpoczyna się podciągiem długości 1 oraz, że różnica długości pomiędzy sąsiednimi podciągami znajdującymi się w jednym rzędzie również wynosi 1. Oznacza to, że czas wyznaczenia wartości

oczekiwanej i wariancji dla pierwszego podciągu w rzędzie jest stały. Jeśli rozpatrzymy sąsiednie podciągi znajdujące się w jednym rzędzie, to wyznaczenie wartości oczekiwanej i wariancji kolejnego podciągu również może zostać zrealizowane w stałym czasie.

Definicja 2.1. *Powiemy, że funkcja L zdefiniowana dla każdego podciągu $s = (t_{l+1}, t_{l+2}, \ldots, t_{l+k})$, jest* obliczeniowo rozdzielcza*, jeżeli spełnia warunek:*

$$L(s) = L(t_{l+1}, t_{l+2}, \ldots, t_{l+k}) = h(L(t_{l+1}, t_{l+2}, \ldots, t_{l+k-1}), L(t_{l+k}))$$

dla pewnej, mającej stałą złożoność obliczeniową, funkcji h.

Wniosek 2.2. *Funkcje*

$$\begin{aligned} M_1(s) &= M_1(t_{l+1}, t_{l+2}, \ldots, t_{l+k}) = t_{l+1} + t_{l+2} + \cdots + t_{l+k}, \\ M_2(s) &= M_2(t_{l+1}, t_{l+2}, \ldots, t_{l+k}) = t_{l+1}^2 + t_{l+2}^2 + \cdots + t_{l+k}^2, \end{aligned}$$

są obliczeniowo rozdzielcze.

Niech R będzie dowolną statystyką, która dla każdego podciągu s spełnia warunek $R(s) = f_s(L_1(s), \ldots, L_m(s))$, gdzie f_s jest funkcją zależną od s i mającą stałą złożoność obliczeniową. Ponieważ $s = (t_{l+1}, t_{l+2}, \ldots, t_{l+k})$ jest jednoznacznie wyznaczone przez l i k, to f_s będziemy również oznaczali przez $f_{l+1,l+k}$.

Data: Ciąg $t = (t_1, \ldots, t_n)$, statystyka R spełniająca warunek $R(s) = f_s(L_1(s), \ldots, L_m(s))$.
Result: Ciąg statystyk $(r_{i,j} = R(s_{i,j}))$ wyznaczonych dla wszystkich możliwych podciągów t.

```
1 for i = 1, 2, ..., n do
2 |   u_{i,i} ← (L_1(s_{i,i}), ..., L_m(s_{i,i}));
3 |   r_{i,i} ← f_{i,i}(u_{i,i});
4 |   for j = i + 1, i + 2, ..., n do
5 |   |   u_{i,j} ← u_{i,j-1} + (L_1(s_{j,j}), ..., L_m(s_{j,j}));
6 |   |   r_{i,j} ← f_{i,j}(u_{i,j});
7 |   end
8 end
```

Algorithm 1: Obliczanie statystyk dla wszystkich podciągów

Twierdzenie 2.3. *Jeśli $s = (t_{l+1}, \ldots, t_{l+k})$ i $R(s) = \frac{1}{k} M_1(s)$, to Algorytm 1 pozwala na wyznaczenie wartości oczekiwanej dla wszystkich podciągów ciągu $(t_1, \ldots, t_n)$ w czasie $O(n^2)$.*

Dowód. Jeśli $s_{i,j} = (t_i, t_{i+1}, \ldots, t_j)$, to definicja funkcji R oznacza, że $L_1(s_{i,j}) = M_1(s_{i,j}) = t_i + \cdots + t_j$ oraz $f_{i,j}(x) = \frac{1}{j-i+1}(x)$. Przy takich oznaczeniach Algorytm 1 przyjmuje postać

for $i = 1, 2, \ldots, n$ **do**
  $u_{i,i} \leftarrow t_i$;
  $r_{i,i} \leftarrow u_{i,i}$;
  for $j = i+1, i+2, \ldots, n$ **do**
    $u_{i,j} \leftarrow u_{i,j-1} + t_j$;
    $r_{i,j} \leftarrow \frac{1}{j-i+1} u_{i,j}$;
  end
end

W pierwszej kolejności wykażemy poprawność tego algorytmu. W tym celu wystarczy udowodnić, że $r_{i,j}$ zawiera średnią arytmetyczną elementów ciągu $(t_i, t_{i+1}, \ldots, t_j)$. Aby to uczynić zauważmy, że dla ustalonego i elementy $u_{i,j}$ spełniają następującą zależność rekurencyjną:

$$\begin{aligned} u_{i,i} &= t_i, \\ u_{i,j} &= u_{i,j-1} + t_j. \end{aligned}$$

Rozwijając tę rekurencję otrzymujemy, że $u_{i,j} = t_i + t_{i+1} + \cdots + t_j$. Ale

$$r_{i,j} = \frac{1}{j-i+1} u_{i,j} = \frac{t_i + t_{i+1} + \cdots + t_j}{j-i+1},$$

co dowodzi, że $r_{i,j}$ jest średnią arytmetyczną ciągu $(t_i, t_{i+1}, \ldots, t_j)$. Aby oszacować złożoność algorytmu zauważmy, że $r_{i,j}$ wyliczane jest na podstawie $u_{i,j}$ w czasie stałym (jedno dzielenie). W związku z tym złożoność algorytmu jest dokładnie taka, jak złożoność wyznaczenia wszystkich wartości $u_{i,j}$. Z zależności rekurencyjnej wynika, że każda kolejna wartość $u_{i,j}$ wyznaczana jest na podstawie $u_{i,j-1}$ w stałym czasie. Zatem złożoność wyznaczenia wszystkich $u_{i,j}$ jest dokładnie taka, jak ich liczba. Ponieważ n jest największą wartością indeksu, to dla ustalonego i mamy dokładnie $n-i+1$ wartości $u_{i,j}$. Zatem moc ciągu $(u_{i,j})$ wynosi dokładnie

$$\begin{aligned} |(u_{i,j})| &= \sum_{i=1}^{n} (n-i+1) = \sum_{i=1}^{n} (n+1) - \sum_{i=1}^{n} i \\ &= n(n+1) - \frac{1}{2} n(n+1) = \frac{1}{2} n(n+1). \end{aligned}$$

W związku z tym złożoność procesu wyznaczania średnich arytmetycznych wszystkich możliwych podciągów wynosi $O(n^2)$, co kończy dowód. □

Twierdzenie 2.4. *Jeśli* $s = (t_{l+1}, \ldots, t_{l+k})$ *i* $R(s) = \frac{1}{k}M_2(s) - \left(\frac{1}{k}M_1(s)\right)^2$, *to Algorytm 1 pozwala na wyznaczenie wariancji dla wszystkich podciągów ciągu* $(t_1, \ldots, t_n)$ *w czasie* $O(n^2)$.

Dowód. Dowód twierdzenia jest bardzo zbliżony, do dowodu Twierdzenia 2.3. Jeśli $s_{i,j} = (t_i, \ldots, t_j)$, to definicja funkcji R oznacza, że

$$\begin{aligned} L_1(s_{i,j}) &= M_1(s_{i,j}) = t_i + \cdots + t_j, \\ L_2(s_{i,j}) &= M_2(s_{i,j}) = t_i^2 + \cdots + t_j^2, \\ f_{i,j}(x,y) &= \frac{1}{j-i+1}(y) - \left(\frac{1}{j-i+1}(x)\right)^2. \end{aligned}$$

Przy takich oznaczeniach Algorytm 1 przyjmuje postać

for $i = 1, 2, \ldots, n$ **do**
  $u_{i,i} \leftarrow (t_i, t_i^2)$;
  $r_{i,i} \leftarrow 0$;
  for $j = i+1, i+2, \ldots, n$ **do**
    $u_{i,j} \leftarrow u_{i,j-1} + (t_j, t_j^2)$;
    $r_{i,j} \leftarrow f_{i,j}(u_{i,j})$;
  end
end

W pierwszej kolejności wykażemy poprawność tego algorytmu. W tym celu wystarczy udowodnić, że $r_{i,j}$ zawiera wariancję elementów ciągu $(t_i, \ldots, t_j)$. Aby to uczynić zauważmy, że dla ustalonego i elementy $u_{i,j}$ spełniają następującą zależność rekurencyjną:

$$\begin{aligned} u_{i,i} &= (t_i, t_i^2), \\ u_{i,j} &= u_{i,j-1} + (t_j, t_j^2). \end{aligned}$$

Rozwijając tę rekurencję otrzymujemy, że $u_{i,j} = (t_i + \cdots + t_j, t_i^2 + \cdots + t_j^2)$. Ale

$$\begin{aligned} r_{i,j} &= f_{i,j}(t_i + \cdots + t_j, t_i^2 + \cdots + t_j^2), \\ &= \frac{t_i^2 + \cdots + t_j^2}{j-i+1} - \left(\frac{t_i + \cdots + t_j}{j-i+1}\right)^2 \end{aligned}$$

co dowodzi, że $r_{i,j}$ jest wariancją ciągu $(t_i, \ldots, t_j)$. Oszacowanie złożoności obliczeniowej można zrobić na dokładnie tej samej zasadzie, jak w przypadku Twierdzenia 2.3. Wystarczy zauważyć, że wyznaczanie kolejnych wartości

$u_{i,j}$, jak i $f_{i,j}(u_{i,j})$ ma stałą złożoność obliczeniową, co oznacza, że złożoność procesu wyznaczania wariancji wszystkich możliwych podciągów wynosi $O(n^2)$, co kończy dowód. □

Wniosek 2.5. *Jeśli funkcja $f_{i,j}$ ma stałą (niezależną do n) złożoność obliczeniową, to złożoność Algorytmu 1 wynosi $O(n^2)$.*

3 Własności numeryczne i stabilność uzyskanego wyniku

O ile wyznaczanie wartości oczekiwanej zgodnie z Twierdzeniem 2.3 nie nastręcza większych problemów jeśli chodzi o precyzję obliczeń, to już zastosowanie Twierdzenia 2.4 w praktyce może nie być proste. Wynika to z faktu, że w przypadku wartości oczekiwanej sumowane są wartości, które w przypadku pomiarów mają zbliżoną wartość. Natomiast wyznaczenie wariancji zaproponowaną metodą wymaga odjęcia od średniej kwadratowej kwadratu średniej arytmetycznej, co może rodzić sporo problemów związanych z precyzją obliczeń. Jeżeli wziąć pod uwagę również fakt, że obliczenia są wykonywane na ciągach, które mogą mieć nawet 10^6 współczynników, to dokładne zbadanie własności numerycznych zaprezentowanych algorytmów jest bardzo istotne.

Jednostka arytmetyczno-logiczna współczesnych procesorów wspiera zazwyczaj zarówno operacje całkowitoliczbowe, jak i zmiennoprzecinkowe (koprocesor matematyczny). Aktualnie 64-bitowe procesory aplikacyjne są w stanie wykonywać bardzo szybkie operacje na: 32 i 64-bitowych liczbach całkowitych oraz 32 i 64-bitowych liczbach zmiennoprzecinkowych (tak zwana pojedyncza i podwójna precyzja). Przy czym liczby zmiennoprzecinkowe reprezentowane są jako

$$S \cdot M \cdot 2^C,$$

gdzie S jest pojedynczym bitem znaku, M jest mantysą, a C jest cechą liczby. W przypadku liczb zmiennoprzecinkowych standard IEEE-754 określa, że dla typu pojedynczej precyzji mantysa ma 23 bity, a cecha jest 8-bitowa. Z kolei mantysa typu podwójnej precyzji ma 52 bity, a jego cecha jest 11-bitowa. Szybkość wykonywania operacji mnożenia liczb na poszczególnych typach zależy od konkretnego rodzaju procesora. Poniżej zestawienie zawierające czasy mnożenia skalarnego wektorów mających 10^8 elementów danego typu:

Typ liczby	Intel Core i5-3360M 2.80GHz	Intel Xeon E3-1240V2 3.40GHz
int32	0.063 s.	0.055 s.
int64	0.072 s.	0.063 s.
single	0.092 s.	0.079 s.
double	0.093 s.	0.082 s.
long double	0.140 s.	0.113 s.

Powyższa tabela jasno pokazuje, że wydajność działania zaproponowanych w poprzedniej części algorytmów byłaby największa, gdyby zaimplementować operacje w oparciu o arytmetykę całkowitoliczbową. Takie podejście może budzić wątpliwości, gdyż wyniki pomiarów są niejako ze swej natury wielkościami zmiennoprzecinkowymi. Jeśli jednak przyjmiemy, że dysponujemy wiedzą o wartości nominalnej mierzonego elementu oraz założymy określoną precyzję pomiaru, to z łatwością będziemy mogli wyliczać odpowiednie statystyki przy użyciu operacji na liczbach całkowitych. Jeżeli na przykład mierzona wielkość ma wartość nominalną $\mu_0 = 125.950$ [mm], a dokładność pomiaru wynosi $\varepsilon_0 = 0.001$ [mm], to wyniki pomiarów μ_i możemy reprezentować jako liczby całkowite m_i, gdzie

$$\mu_i = \mu_0 + m_i\varepsilon_0$$

jest rzeczywiście zmierzoną wartością. Wyznaczając m_i z powyższej zależności otrzymujemy

$$m_i = \frac{\mu_i - \mu_0}{\varepsilon_0}.$$

Ogromną zaletą takiej reprezentacji jest to, że wyznaczanie wartości oczekiwanej i wariancji może być zrealizowane bez błędów zaokrągleń. Niektórych z nich nie da się uniknąć w przypadku reprezentacji zmiennoprzecinkowej, gdyż jeśli na przykład urządzenie pomiarowe daje wyniki z dokładnością do 0.1, to reprezentacja zmiennoprzecinkowa liczby $\frac{1}{10}$ w systemie dwójkowym jest nieskończona i ma postać:

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \cdots = (0.0001100110011(0011))_2.$$

Problemów tego rodzaju możemy uniknąć przechodząc do reprezentacji całkowitoliczbowej. Warunkiem jej poprawnego działania jest jednak precyzyjna kontrola wartości, przez którą przetwarzane liczby całkowite zostały wcześniej pomnożone. Takie podejście nie sprawdzi się w przypadku zbiorów przypadkowych liczb, ale jeśli chodzi o wyniki pomiarów, to będzie rozwiązaniem idealnym.

W dalszej części naszych rozważań skupimy się na wyznaczeniu liczby bitów mantysy niezbędnych do poprawnego wyliczenia wartości oczekiwanej i wariancji. Będziemy przy tym zakładali, że operujemy typem całkowitoliczbowym czyli, że liczba bitów cechy jest zerowa. Przy takich założeniach mamy następujące twierdzenie dotyczące algorytmów zawartych w dowodach twierdzeń 2.3 i 2.4:

Twierdzenie 3.1. *Niech μ_0 będzie wartością nominalną mierzonej wielkości, a ε_0 dokładnością, z jaką dokonujemy pomiaru. Załóżmy ponadto, że ustalony jest zakres pomiarowy i wyniki mieszczą się w przedziale $[\mu_0 - r_0, \mu_0 + r_0)$. Jeżeli n jest liczbą pomiarów, $A_n(\varepsilon_0, r_0)$ jest liczbą bitów typu całkowitoliczbowego niezbędną do poprawnego wyznaczenia wartości oczekiwanej, a $V_n(\varepsilon_0, r_0)$ jest liczbą bitów typu całkowitoliczbowego niezbędną do poprawnego wyznaczenia wariancji, to:*

$$\begin{aligned} A_n(\varepsilon_0, r_0) &= \left\lfloor \log_2 \left(\frac{2nr_0}{\varepsilon_0} \right) \right\rfloor + 1, \\ V_n(\varepsilon_0, r_0) &= \left\lfloor \log_2 \left(\frac{nr_0^2}{\varepsilon_0^2} \right) \right\rfloor + 1. \end{aligned}$$

Dowód. Dla dowodu twierdzenia wystarczy wykazać, że największe liczby występujące w algorytmach twierdzeń 2.3 i 2.4 mieszczą się w rejestrach bitowych o długościach odpowiednio $A_n(\varepsilon_0, r_0)$ i $V_n(\varepsilon_0, r_0)$. Zauważmy, że jeśli pomiaru dokonujemy z dokładnością ε_0, to na przedziale $[\mu_0 - r_0, \mu_0 + r_0)$ mamy $\left\lfloor \frac{2r_0}{\varepsilon_0} \right\rfloor$ możliwych wyników pomiaru. Oznacza to, że wyniki μ_i mogą być reprezentowane jako liczby całkowite $m_i = \frac{\mu_i - \mu_0}{\varepsilon_0}$ spełniające nierówność

$$-\frac{r_0}{\varepsilon_0} \leq m_i < \frac{r_0}{\varepsilon_0}.$$

Największą wartością występującą w dowodzie twierdzenia 2.3 jest $u_{1,n} = t_1 + \cdots + t_n = m_1 + \ldots m_n$. Oznacza to, że $u_{1,n}$ spełnia następującą nierówność

$$-\frac{nr_0}{\varepsilon_0} \leq u_{1,n} < \frac{nr_0}{\varepsilon_0}.$$

Przypominając, że liczba bitów niezbędna do reprezentowania całkowitej liczby nieujemnej m wyraża się wzorem $\lfloor \log_2 m \rfloor + 1$ i uwzględniając bit znaku otrzymujemy

$$A_n(\varepsilon_0, r_0) = \left\lfloor 1 + \log_2 \left(\frac{nr_0}{\varepsilon_0} \right) \right\rfloor + 1 = \left\lfloor \log_2 \left(\frac{2nr_0}{\varepsilon_0} \right) \right\rfloor + 1.$$

Z kolei największą wartością występującą w dowodzie twierdzenia 2.4 jest druga współrzędna $(u_{1,n})_2 = t_1^2 + \cdots + t_n^2 = m_1^2 + \ldots m_n^2$ zmiennej $u_{1,n}$. Uwzględniając ograniczenie na wielkość m_i otrzymujemy, że $(u_{1,n})_2$ spełnia następującą nierówność

$$0 \leq (u_{1,n})_2 < \frac{nr_0^2}{\varepsilon_0^2}.$$

Otrzymujemy zatem, że

$$V_n(\varepsilon_0, r_0) = \left\lfloor \log_2 \left(\frac{nr_0^2}{\varepsilon_0^2} \right) \right\rfloor + 1,$$

co kończy dowód. □

4 Praktyczna implementacja algorytmu i jej wyniki

Na potrzeby analizy proponowanego algorytmu oraz wyboru odpowiedniego typu danych opracowano implementację w języku C++. Jej pierwszym zadaniem było określenie szybkości obliczeń na dostępnych typach danych. Analizie poddano pięć typów liczbowych:

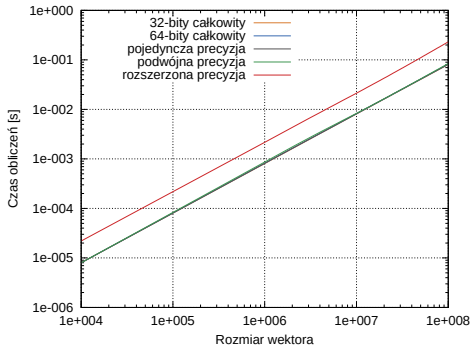
1. 32-bitowy typ całkowitoliczbowy `int32_t`,
2. 64-bitowy typ całkowitoliczbowy `int64_t`,
3. 32-bitowy typ zmiennoprzecinkowy z 23-bitową mantysą `float` (pojedyncza precyzja),
4. 64-bitowy typ zmiennoprzecinkowy z 53-bitową mantysą `double` (podwójna precyzja),
5. 80-bitowy typ zmiennoprzecinkowy z 63-bitową mantysą `long double` (rozszerzona precyzja).

Ze względu na zastosowanie do analizy dużych zbiorów danych zainteresowanie nasze zostało skupione na typach `int64_t` i `long double`, gdyż tylko one dają realną możliwość dokładnego wyznaczania wartości wariancji dla próbek liczących więcej niż 10^6 elementów i reprezentowanych z 16-bitową precyzją. Jednak w celu porównania wszystkich możliwości, każdy

	Czas wykonania operacji w [ms] dla wektora mającego 10^6 elementów				
Operacja	`int32_t`	`int64_t`	`float`	`double`	`long double`
Min/Max	0.806	0.814	0.817	0.865	2.169
Wartość oczekiwana	0.416	0.476	0.803	0.805	0.954
Wariancja	1.078	1.155	1.610	1.618	1.980
Wszystko	1.890	1.969	2.404	2.445	4.115

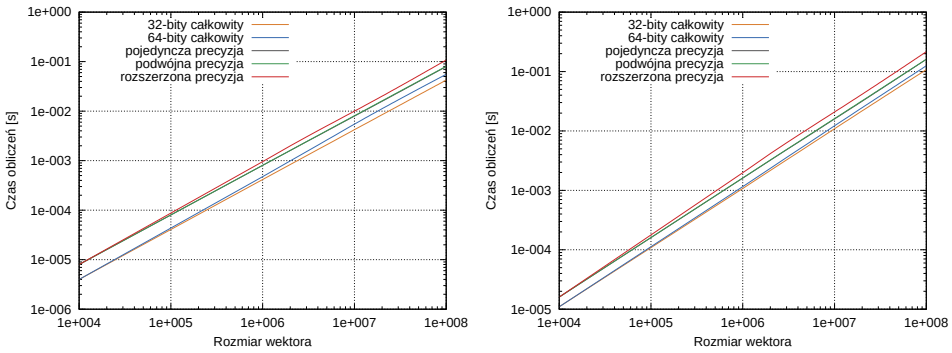
Tablica 1: Czasy wyznaczania podstawowych statystyk wektora mającego 10^6 elementów. Pomiary wykonane dla poszczególnych typów danych.

z wymienionych wcześniej typów danych został poddany analizie wydajnościowej. Tablica 1 zawiera czas wykonania operacji: wyznaczania ekstremów, wyznaczania wartości oczekiwanej, wyznaczania wariancji i jednoczesnego wyznaczania wszystkich wymienionych statystyk. W tabeli umieszczono czasy jedynie dla ciągów mających 10^6 elementów. Informacje o szybkości działania poszczególnych typów dla większego zakresu można odczytać z wykresów umieszczonych na Rysunkach 1, 2 i 3.



Rysunek 1: Czas wyznaczenia minimum i maksimum (obie osie ze skalą logarytmiczną).

Analiza czasów działania pokazuje, że typ `int64_t` jest ponad dwa razy szybszy względem `long double`. Ze względu na charakter prowadzonych przez nas obliczeń (wyznaczanie ekstremów, wartości oczekiwanej i wariancji) oba typy gwarantują praktycznie identyczną precyzję. Dodatkowo typ całkowitoliczbowy pozwala na uniknięcie błędów zaokrągleń, gdy podstawą

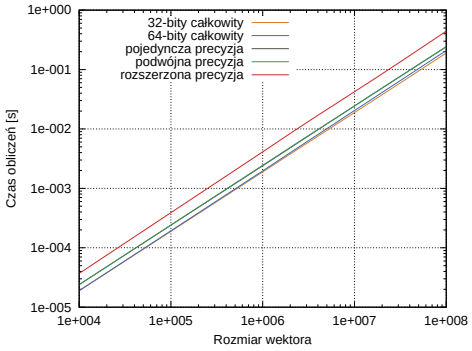


Rysunek 2: Czas wyznaczenia wartości oczekiwanej (lewy) i wariancji (prawy) (obie osie ze skalą logarytmiczną).

dokładności pomiarów są wielokrotności potęgi liczby 10. W Tablicy 2 umieszczono pomiary czasu działania proponowanego algorytmu w zestawieniu ze standardowym algorytmem, który wyznacza statystyki niezależnie dla każdego z podciągów. Dokładniejszy obraz szybkości działania algorytmów na poszczególnych typach obliczeniowych ilustruje Rysunek 4. Widać na nim wyraźnie różnicę w złożoności obu algorytmów. Wraz ze wzrostem rozmiaru analizowanych danych czas potrzebny do wyznaczenia podstawowych statystyk bardzo szybko rośnie. Jeżeli analizie zostałby poddany ciąg mający około 10^6 elementów, to szacowany czas działania proponowanego algorytmu wynosi około 30 godzin. Natomiast niezoptymalizowany algorytm tę samą analizę przeprowadzałby niemal 4 lata. Podstawowym warunkiem rzetelności zaproponowanego algorytmu jest taki dobór typu obliczeniowego, który pozwoli na bezbłędne wyznaczanie sum i sum kwadratów dla dużej liczby elementów. Dlatego też zaproponowane zostało wykorzystanie typów całkowitoliczbowych zamiast zmiennoprzecinkowych.

Na Rysunku 5 przedstawiono porównanie szybkości działania proponowanego algorytmu dla dwóch rozważanych typów danych: `long double` i `int64_t`. Należy podkreślić, że przewaga szybkości działania typu całkowitoliczbowego nad zmiennoprzecinkowym uległa znaczącemu zmniejszeniu. Można wskazać cztery przyczyny takiego stanu rzeczy:

1. duża liczba krótkich podciągów, dla których wyznaczane są statystyki,
2. konieczność przechowywania ogromnej liczby wyznaczonych statystyk, co przekłada się na dużą liczbę operacji na pamięci,



Rysunek 3: Czas jednoczesnego wyznaczenia wartości oczekiwanej, wariancji, minimum i maksimum (obie osie ze skalą logarytmiczną).

3. większa liczba instrukcji warunkowych w pętlach wyznaczających kolejne statystyki (więcej czasu procesor spędza na analizie warunków niż na faktycznych obliczeniach),

4. konieczność wykonywania dzieleń zmiennoprzecinkowych podczas wyznaczania średniej i wariancji (są one tak kosztowne, że praktycznie niwelują zysk z zastosowania całkowitoliczbowego typu podstawowego).

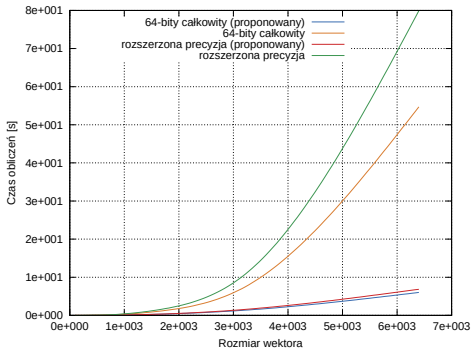
Wstępna analiza implementacji wskazuje, że czwarta przyczyna ma największy wpływ na tak drastyczne zmniejszenie się różnicy pomiędzy czasami obliczeń dla typów `int64_t` i `long double`. Oczywiście powstaje pytanie, jak czas działania będzie kształtował się dla ciągów mających 10^5 lub 10^6 elementów. Innym istotnym zagadnieniem jest optymalizacja dostępów do pamięci lub też takie skonstruowanie algorytmu, aby nie zapamiętywał statystyk, a jedynie miejsca, które powinny zostać poddane dokładniejszej analizie. Na Rysunku 6 zaprezentowano stosunek szybkości działania algorytmu wolniejszego do szybkości działania algorytmu proponowanego dla dwóch rozważanych typów obliczeniowych. Widać z niego, że typ zmiennoprzecinkowy ma ten stosunek lepszy od typu całkowitoliczbowego. Może się więc okazać, że dla odpowiednio długich ciągów bardziej opłacalne będzie wykorzystanie jednak typu `long double`.

Rozmiar wektora	Liczba podciągów	Czas wyznaczenia statystyk w [ms] `int64_t` prop.	std.	`long double` prop.	std.
100	5050	0.54	0.84	0.67	1.11
200	20100	2.27	4.20	3.04	5.88
400	80200	11.49	25.40	14.08	35.00
800	320400	56.44	156.10	67.73	206.20
1600	1280800	285.00	1031.00	301.00	1407.00
3200	5121600	1344.00	7360.00	1547.00	10610.00
6400	20483200	6016.00	54706.00	6844.00	79926.00

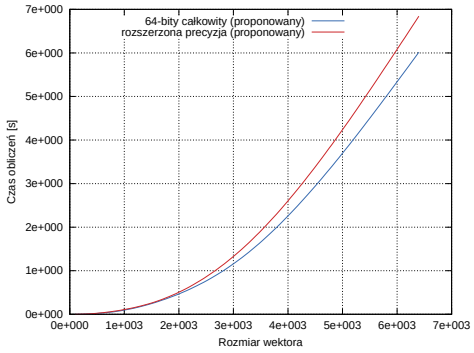
Tablica 2: Porównanie czasów działania algorytmu proponowanego (prop.) i standardowego (std.) dla dwóch typów obliczeniowych: 64-bitowego typu całkowitego i 80-bitowego typu rozszerzonego z 63-bitową mantysą.

5 Podsumowanie

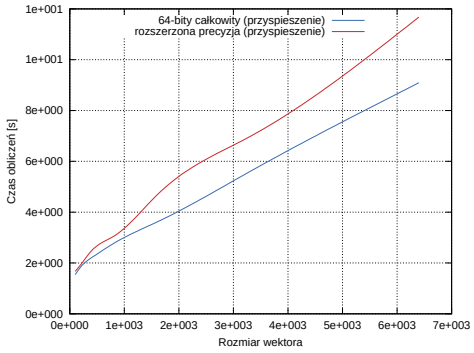
W pierwszej części artykułu przedstawiono algorytmy służące do szybkiej analizy podstawowych własności statystycznych wszystkich podciągów pewnego ciągu liczbowego. Głównym zastosowaniem zaprezentowanego algorytmu ma być wstępna weryfikacja, które obszary wewnątrz całego ciągu mają pewne nieoczekiwane właściwości. Obszary te będą następnie poddawane dalszej (bardziej wnikliwej) analizie. Złożoność obliczeniowa zaprezentowanego algorytmu wynosi $O(n^2)$, gdzie n jest liczbą elementów ciągu. Jest to znaczne przyspieszenie względem klasycznej metody polegającej na systematycznym wyznaczaniu statystyk dla każdego podciągu z osobna, którego złożoność wynosi $O(n^3)$. Porównanie działania obu algorytmów zostało zaprezentowane w Tablicy 2 oraz na Rysunkach 4 i 6. W kolejnej części zaprezentowane zostały metody prowadzenia obliczeń na danych pomiarowych przy wykorzystaniu zarówno reprezentacji zmiennoprzecinkowej, jak i całkowitoliczbowej. W Twierdzeniu 3.1 przedstawiono warunki, jakie musi spełniać typ liczbowy (jego mantysa), aby mógł być wykorzystany do precyzyjnego wyznaczenia takich podstawowych statystyk, jak wartość oczekiwana i wariancja.



Rysunek 4: Czas wyznaczenia wartości oczekiwanej i wariancji dla wszystkich możliwych podciągów.



Rysunek 5: Czas wyznaczenia wartości oczekiwanej i wariancji dla wszystkich możliwych podciągów (tylko 64-bitowy typ całkowity i 80-bitowy typ rozszerzony z 63-bitową mantysą).



Rysunek 6: Przyspieszenie obliczeń dawane przez proponowany algorytm (tylko 64-bitowy typ całkowity i 80-bitowy typ rozszerzony z 63-bitową mantysą).