Metody numeryczne

Zagadnienia wstępne

P. F. Góra

https://zfs.fais.uj.edu.pl/pawel_gora

8 października 2024

Źródła błędów numerycznych

Wyniki obliczeń numerycznych obarczone są błędami. Ich najważniejszymi źródłami są

1. **Błędy grube**, pomyłki — zawinione przez człowieka. Czasami trudno je wyeliminować, ale teoretycznie wszystkie można usunąć, dlatego też, zalecając ostrożność i staranność autorom programów numerycznych, w analizie algorytmów pomijamy wpływ tych błędów.
2. **Błędy algorytmu**, wynikające z zastąpienia "idealnego" problemu matematycznego przez jakieś przybliżenie, na przykład zastąpienie zagadnienia obliczania całki oznaczonej przez obliczanie skończonego ciągu sum. W analizie numerycznej staramy się oszacować wpływ takich błędów. Algorytm, którego błędu nie umiemy oszacować, można uznać za bezwartościowy.

Dla niektórych problemów istnieją tak zwane algorytmy dokładne, które dałyby matematycznie ścisłe wyniki, gdyby można było pominąć...

3. Konsekwencje **błędów zaokrąglenia**, wynikających z tego, że nie wszystkie liczby dają się reprezentować na skończonych komputerach w sposób dokładny, a zatem obliczenia prowadzone są *ze skończoną precyzją (dokładnością)*. Tych błędów, poza bardzo szczególnymi przypadkami, nie daje się całkowicie wyeliminować, należy natomiast umieć szacować ich wpływ oraz go minimalizować.

W ogólnym przypadku wpływ błędu algorytmu i błędu zaokrąglenia mogą się na siebie nakładać i wzajemnie wzmacniać.

Błąd zaokrąglenia

Sposoby reprezentacji liczb całkowitych i rzeczywistych — patrz wykład z Teoretycznych Podstaw Informatyki.

W sposób *ścisły* można reprezentować w komputerze tylko liczby całkowite (z pewnego zakresu) oraz liczby wymierne, posiadające *skończone* rozwinięcia binarne (z pewnego zakresu). Wszystkie inne liczby można reprezentować tylko w sposób przybliżony. Są one zatem obarczone pewnym błędem, zwanym *błędem zaokrąglenia*.

Rozwinięcia binarne

Niech liczba $x \in (0, 1)$. Wówczas jej rozwinięciem binarnym jest

$$x = \sum_{i=1}^{\infty} a_i \cdot 2^{-i} \tag{1}$$

gdzie współczynniki a_i mogą przybierać tylko wartości $\{0, 1\}$.

Przykład

Znajdźmy rozwinięcie binarne liczby $\frac{1}{5}$. Mamy

$$\begin{aligned}
\frac{1}{5} &= \frac{1}{8}+\frac{1}{5}-\frac{1}{8}=\frac{1}{8}+\frac{3}{40}=\frac{1}{8}+\frac{1}{8}\cdot\frac{3}{5} \\
&= \frac{1}{8}+\frac{1}{8}\left(\frac{1}{2}+\frac{3}{5}-\frac{1}{2}\right)=\frac{1}{8}+\frac{1}{16}+\frac{1}{8}\cdot\frac{6-5}{2\cdot 5}=\frac{1}{8}+\frac{1}{16}+\frac{1}{16}\cdot\frac{1}{5} \\
&= \frac{1}{8}+\frac{1}{16}+\frac{1}{16}\left(\frac{1}{8}+\frac{1}{16}+\frac{1}{16}\cdot\frac{1}{5}\right) \\
&= \frac{1}{8}+\frac{1}{16}+\frac{1}{128}+\frac{1}{256}+\frac{1}{256}\cdot\frac{1}{5} \\
&= \frac{1}{8}+\frac{1}{16}+\frac{1}{128}+\frac{1}{256}+\frac{1}{2048}+\frac{1}{4096}+\frac{1}{32768}+\frac{1}{65536} \\
&\quad +\frac{1}{65536}\cdot\frac{1}{5} \\
&= 2^{-3}+2^{-4}+2^{-7}+2^{-8}+2^{-11}+2^{-12}+2^{-15}+2^{-16}+\ldots \\
&= 0.001100110011(0011)_2 \qquad (2)
\end{aligned}$$

Otrzymaliśmy nieskończone, okresowe rozwinięcie binarne.

Analogicznie, ponieważ $\frac{1}{10} = \frac{1}{2} \cdot \frac{1}{5}$, a w rozwinięciach binarnych dzielenie przez 2 oznacza przesunięcie wszystkiego o jedną pozycję w prawo,

$$\frac{1}{10} = 0.0001100110011(0011)_2 \tag{3}$$

Widać stąd, że "naturalne" dla osób posługujących się systemem dziesiętnym ułamki $\frac{1}{10}$, $\frac{1}{100}$ itd *nie mogą być* ***dokładnie*** *reprezentowane* w pamięci. Komputer zapamiętuje tylko ich skończone przybliżenia, obarczone błędem zaokrąglenia. Po bardzo wielu krokach błąd ten może się skomasować do czegoś, co istotnie wpłynie na wynik obliczeń.

Zamiast więc iterować z krokiem $\frac{1}{10}$, iterujmy z krokiem $\frac{1}{8}$. Zamiast z $\frac{1}{100}$ weźmy $\frac{1}{128}$ itd.

Cyfry znaczące

Niech x będzie liczbą rzeczywistą, mającą ogólnie nieskończone rozwinięcie dziesiętne. Cyfry tego rozwinięcia numerujemy w sposób "naturalny": cyfra jedności ma numer zero, cyfra dziesiątek ma numer jeden, cyfra setek ma numer dwa itd. Cyfry części ułamkowej rozwinięcia dziesiętnego mają numery ujemne.

Liczba x jest poprawnie zaokrąglona na d-ej pozycji do liczby, którą oznaczamy $x^{(d)}$, jeśli błąd zaokrąglenia ε jest taki, że

$$|\varepsilon| = \left|x - x^{(d)}\right| \leqslant \frac{1}{2} \cdot 10^d \,. \tag{4}$$

Na przykład jeśli $x = 6.743996666\ldots$, $x^{(-3)} = 6.744$, $x^{(-7)} = 6.7439967$.

Jeśli $\bar{x}$ jest przybliżeniem dokładnej wartości x, to k-tą cyfrę dziesiętną liczby $\bar{x}$ nazwiemy *znaczącą*, jeśli

$$|x - \bar{x}| \leqslant \frac{1}{2} \cdot 10^k \tag{5}$$

(części ułamkowe rozwinięcia mają numery ujemne!). Wynika stąd, że każda cyfra poprawnie zaokrąglonej liczby, począwszy od pierwszej cyfry różnej od zera, jest znacząca. Liczba cyfr znaczących jest pewną miarą błędu zaokrąglenia.

Jeżeli w wyniku obliczeń otrzymamy jakąś wielkość $\bar{x}$ z dokładnością do ε, możemy jedynie stwierdzić, że *"prawdziwa"* wartość obliczanej wielkości leży w przedziale $[\bar{x} - \frac{1}{2}\varepsilon, \bar{x} + \frac{1}{2}\varepsilon]$.

Przykład

Obliczając numerycznie jakąś wielkość, otrzymaliśmy wynik 2.3458, z (domyślną) dokładnością 10^{-4}. Oznacza to, że — jeżeli możemy pominąć błędy grube i błędy algorytmu — "prawdziwa" wartość obliczanej wielkości zawiera się w przedziale $[2.34575, 2.34585]$.

Obowiązujące zasady zaokrąglania

- Jeśli pierwszą odrzucaną cyfrą jest 0, nie zaokrąglamy. $1.2309 \simeq 1.23$.
- Jeśli pierwszą odrzucaną cyfrą jest 1,2,3,4, zaokrąglamy w dół. $1.234 \simeq 1.23$.
- Jeśli pierwszą odrzucaną cyfrą jest 6,7,8,9, zaokrąglamy w górę. $1.236 \simeq 1.24$.
- Jeśli pierwszą odrzucaną cyfrą jest 5, zaokrąglamy do *najbliższej parzystej*. $1.235 \simeq 1.24$, ale także $1.245 \simeq 1.24$. Bierze się to stąd, że przyjęcie zasady "5 zawsze w górę" lub "5 zawsze w dół" wprowadzałoby *systematyczny* błąd do obliczeń.

Propagacja błędu

Niech $\bar{x}$ będzie poprawnie zaokrąglonym do d przybliżeniem dokładnej liczby x. Można powiedzieć, że $x = \bar{x} + \varepsilon$, gdzie ε jest liczbą losową o rozkładzie jednostajnym w przedziale $\left[-\frac{1}{2}10^d, \frac{1}{2}10^d\right]$. Weźmy teraz dwie liczby $x = \bar{x} + \varepsilon_x$ oraz $y = \bar{y} + \varepsilon_y$. Co można powiedzieć o błędach powstałych w wyniku elementarnych obliczeń artymetycznych?

$$x + y = \bar{x} + \bar{y} + \underbrace{\varepsilon_x + \varepsilon_y}_{\varepsilon_+} . \tag{6}$$

Liczba ε_+ jest liczbą losową. Jeżeli obie liczby ε_x, ε_y mają rozkład jednostajny na przedziale $\left[-\frac{1}{2}10^d, \frac{1}{2}10^d\right]$, liczba ε_+ ma rozkład trójkątny na przedziale $\left[-10^d, 10^d\right]$.

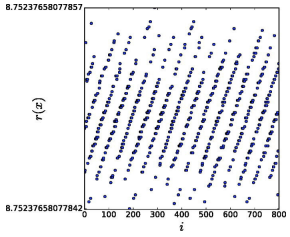
Jeśli będziemy sumować $N \gg 1$ liczb o błędach zaokrąglenia pochodzących z tego samego rozkładu, błąd sumy będzie dążył do rozkładu normalnego o szerokości proporcjonalnej do $\sqrt{N}\,10^d$.

Uwaga!

W przypadku bardziej złożonych operacji błędy nie muszą mieć (i na ogół nie mają) charakteru nieskorelowanych liczb losowych. Rozpatrzmy obliczanie wartości funkcji*

$$r(x) = \frac{4x^4 - 59x^3 + 324x^2 - 751x + 622}{x^4 - 14x^3 + 72x^2 - 151x + 112} \tag{7}$$

dla $x = 1.606 + 2^{-52} i\,, i = 1, 2, \ldots$.



*A. Gezerlis, M. Williams, Am. J. Phys. **89**, 51, 2021

Błędy, choć bardzo małe, nie układają się losowo (równomiernie), tylko tworzą wyraźny wzorzec[†]. W złożonych obliczeniach błędy są *skorelowane*: wartość błędu w kroku i nie jest niezależna od wartości błędu w krokach $i-1, i-2, \ldots$. Może to negatywnie wpływać na ich wzajemne kasowanie się i dodatkowo numerycznie pogarszać wynik obliczeń.

Wynika stąd praktyczny wniosek, że błędy, o ile to możliwe, należy eliminować na jak wcześniejszych etapach obliczeń.

[†]Błędy w obliczaniu funkcji (7) stanowią *kiepski* generator liczb pseudolosowych ☺

Wróćmy do przypadku prostego, gdy trzeba oszacować błąd w pojedynczej operacji arytmetycznej.

W wypadku *mnożenia*,

$$x \cdot y \approx \bar{x} \cdot \bar{y} + \bar{x}\varepsilon_y + \bar{y}\varepsilon_x \,. \tag{8}$$

Jeżeli $|\bar{x}| \gg |\bar{y}|$ lub $|\bar{y}| \gg |\bar{x}|$, może to spowodować znaczny wzrost błędu (niewielki błąd multiplikuje się).

W wypadku *dzielenia*,

$$\frac{x}{y} \simeq \frac{\bar{x}}{\bar{y}} + \frac{1}{\bar{y}}\varepsilon_x + \frac{\bar{x}}{\bar{y}^2}\varepsilon_y \,. \tag{9}$$

Jeżeli $|\bar{y}| \ll |\bar{x}|$, błąd dzielenia może być bardzo duży! Dzielenie przez (względnie) małe liczby obarczone błędem, może powodować pojawienie się **znacznego** błędu ilorazu.

Wpływ błędów zaokrąglenia

Błędy zaokrąglenia — fakt, że na komputerach pracujemy w arytmetyce ze skończoną dokładnością — mają wpływ na prowadzone obliczenia. Wynik może zależeć od kolejności przeprowadzanych działań: dodawanie "na komputerze" nie jest łączne!

Przykład: Przypuśćmy, że prowadząc obliczenia z dokładnością do czterech cyfr znaczących chcemy znaleźć wartość sumy

$$1.000 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 + 0.0001 \quad (10)$$

Zaczynamy sumować od lewej. Suma dwu pierwszych składników wynosi 1.0001. Ta liczba ma *pięć* cyfr znaczących, więc zostanie zaokrąglona do czterech cyfr znaczących. I tak okazuje się, że w przyjętej dokładności, $1.000 + 0.0001 = 1.000$. Widać zatem, że przy tym sposobie prowadzenia obliczeń, wartość *całej* sumy (10) wynosi 1.000.

Sumujmy teraz od prawej. $0.0001+0.0001 = 0.0002$. $0.0002+0.0001 = 0.0003$ i tak dalej. Suma dziesięciu pierwszych (od prawej) składników wynosi 0.0010. Gdy teraz dodamy tę wielkość do składnika ostatniego od prawej, otrzymamy 1.001. Ta liczba ma cztery cyfry znaczące, co mieści się w przyjętej dokładności i wyniku nie trzeba zaokrąglać.

Inny przykład

Rozważmy ciąg zadany przepisem

$$\begin{cases} x_{n+1} &= 4x_n - 1 \\ x_0 &= \frac{1}{3}\,. \end{cases} \tag{11}$$

Jak łatwo sprawdzić, ciąg (11) jest ciągiem stałym, którego wszystkie wyrazy są równe $\frac{1}{3}$. Co jednak się stanie, jeśli wyrazy tego ciągu będziemy *obliczali* posługując się arytmetyką przybliżoną, zachowując osiem cyfr znaczących?

Mamy zatem

$$x_0 = 0.33333333 \quad (12a)$$

$$4x_0 = 1.33333332 \quad (12b)$$

Ostatnia cyfra w powyższym wyrażeniu byłaby *dziewiątą* cyfrą znaczącą – nie mamy miejsca na jej przechowywanie, a więc musimy ją odrzucić. Zatem

$$4x_0 = 1.33333330 \quad (12c)$$

$$x_1 = 0.33333330 \quad (12d)$$

W podobny sposób wyliczamy $x_2 = 0.33333280$. Wyniki kolejnych iteracji przedstawia poniższa tabela:

n	x_n
0	0.33333333
1	0.3333333
2	0.3333328
3	0.3333312
4	0.3333248
5	0.3332992
6	0.3331968
7	0.3327872
8	0.3311488
9	0.3245952
10	0.2983808
11	0.1935232
12	−0.2259072
13	−1.9036288
14	−8.6145152
15	−35.458061
16	−142.83224

Jak widzimy, **w wyniku prowadzenia obliczeń ze skończoną precyzją**, ciąg stały zamienił się w ciąg monotonicznie rozbieżny do $-\infty$. Gdybyśmy

prowadzili obliczenia z większą – ale wciąż skończoną – precyzją, rezultat byłby taki sam, choć liczba stanów “przejściowych”, gdy wyrazy ciągu wciąż są bliskie ścisłej wartości $\frac{1}{3}$, zwiększyłaby się.

Reguła sumacyjna Kahana

Przy "zwykłym" sumowaniu n składników błąd numeryczny, w najgorszym wypadku, jest proporcjonalny do n, a sredni błąd kwadratowy przy sumowaniu składników losowych rośnie jak $\sqrt{n}$‡. Kahan zaproponował algorytm, w którym trzyma się dodatkową zmienną akumulującą małe błędy. Dzięki temu błąd w najgorszym przypadku praktycznie nie zależy od n.

```
function KahanSum(input)
    var sum = 0.0
    var c = 0.0

    for i = 1 to input.length do
        var y = input[i] - c
        var t = sum + y
        c = (t - sum) - y
        sum = t
    next i
    return sum
```

‡Konsekwencja Centralnego Twierdzenia Granicznego.

W arytmetyce dokładnej powinno być $c \equiv 0$ — ale w arytmetyce o skończonej dokładności tak nie jest. Jest jednak **niesłychanie** ważne, aby wyrażenie w nawiasach

```
c = (t - sum) - y
```

było obliczane *przed* kolejnym odejmowaniem. Trzeba pilnować, aby kompilator/interpreter/optymalizator nie potraktował tych nawiasów jako zbędnych.

Inne podejście

Załóżmy, że mamy zsumować $n \gg 1$ liczb dodatnich, które mogą *znacznie* różnić się co do rzędów wielkości. Wówczas należy te liczby posortować i sumować od najmniejszej do największej. Czas działania takiego algorytmu wydłuża się do $O(n \log n)$.

Jeżeli liczby mogą mieć różne znaki, dzielimy je na dwie kohorty: dodatnią i ujemną. Sortujemy, a po posortowaniu sumujemy każdą kohortę poczynając od liczb o najmniejszym module. Na końcu dodajemy obie sumy, w ten sposób kombinujemy znaki tylko raz.

