

Analiza Numeryczna

Skrypt z wykładów

Semestr zimowy 2023/2024

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Spis treści

1	2
2 Arytmetyka zmiennopozycyjna, podstawy teorii błędów	3
2.1 Błąd bezwzględny i błąd wzg̊elny	3
2.2 Reprezentacja liczb w komputerze	3
2.3 Zjawisko utraty cyfr znaczących	7

Wykład 1

Wykład 2 Arytmetyka zmiennopozycyjna, podstawy teorii błędów

2.1 Błąd bezwzględny i błąd względny

Weźmy liczby:

$$x = 1.23456789 \quad \tilde{x} = 1.2345679 \\ y = 10^{50} + 1 \quad \tilde{y} = 10^{50}$$

Wtedy błąd bezwzględny to:

$$|x - \tilde{x}| = 10^{-8} \quad |y - \tilde{y}| = 1$$

Błąd względny to:

$$\frac{|x - \tilde{x}|}{|x|} = 0.8 \cdot 10^{-8} \quad \frac{|y - \tilde{y}|}{|y|} = 10^{-50}$$

Błąd względny jest lepszą miarą błędu.

Dodatkowo zdefiniujmy liczbę cyfr dokładnych:

$$acc(v, \tilde{v}) = -\log_{10} \left(\left| 1 - \frac{\tilde{v}}{v} \right| \right)$$

Wtedy:

$$acc(x, \tilde{x}) \approx 8.091 \quad acc(y, \tilde{y}) = 50$$

2.2 Reprezentacja liczb w komputerze

Potrafimy reprezentować wszystkie liczby całkowite z pewnego zakresu, ale nie potrafimy reprezentować wszystkich liczb rzeczywistych. Dlatego musimy wybrać pewną reprezentację, która będzie przybliżać liczby rzeczywiste.

a) $l \in \mathbb{Z}$,

$$l = \pm \sum_{i=0}^n e_i 2^i, \quad e_i \in \{0, 1\}, \quad e_n = 1$$

Jeśli $n < d$ to OK, a jeśli $n \geq d$ to przepełnienie.

b) $x \in \mathbb{R} \setminus \{0\}$

TW. Dla każdej liczby rzeczywistej $x \neq 0$ istnieje trójka:

$$m \in \left[\frac{1}{2}, 1 \right) \quad (\text{mantysa})$$

$$c \in \mathbb{Z} \quad (\text{cecha})$$

$$s \in \{+1, -1\} \quad (\text{znak liczby})$$

dla których

$$x = s \cdot m \cdot 2^c$$

Trójka (s, m, c) jest wyznaczona jednoznacznie.

Reprezentacja mantysy:

$$m \in \left[\frac{1}{2}, 1 \right), \quad m = \sum_{i=1}^{\infty} e_{-i} 2^{-i}, \quad e_{-i} \in \{0, 1\}, \quad e_{-1} = 1$$

Sposoby zaokrąglania mantysy:

i) obcięcie

$$m_t^c := \sum_{i=1}^t e_{-i} 2^{-i}$$

ii) zaokrąglanie symetryczne

$$m_t^r := \sum_{i=1}^t e_{-i} 2^{-i} + e_{-(t+1)} 2^{-t}$$

Model zapisu liczby:

$$\underbrace{[\pm]}_{t \text{ bitów}} \underbrace{[\text{bity mantysy}]}_{(d-t) \text{ bitów}} \quad \underbrace{[\text{cecha ze znakiem}]}_{(d-t) \text{ bitów}}$$

Łącznie: $d + 1$ bitów na liczbę rzeczywistą ze znakiem.

Ten model reprezentacji jest teoretyczny. W praktyce stosujemy standard IEEE 754.

TW.

$$|m - m_t^c| \leq 2^{-t}, \quad |m - m_t^r| \leq \frac{1}{2} \cdot 2^{-t}$$

Reprezentacja zmiennopozycyjna liczby rzeczywistej $x \neq 0$:

$$x \approx \text{chop}(x) := s \cdot m_t^c \cdot 2^c \quad \text{albo} \quad x \approx \text{rd}(x) := s \cdot m_t^r \cdot 2^c$$

Pytanie: Które liczby rzeczywiste można dokładnie reprezentować w komputerze? Jaką one mają postać?

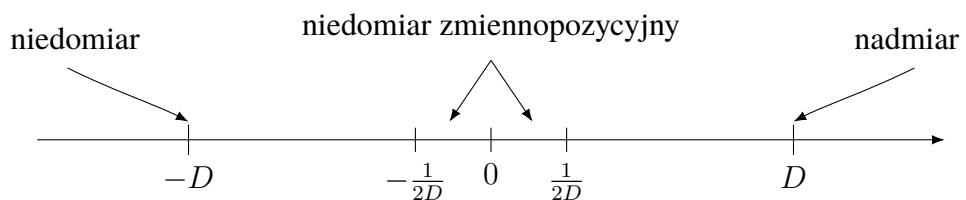
TW.

$$\left| \frac{\text{chop}(x) - x}{x} \right| \leq 2 \cdot 2^{-t}, \quad \left| \frac{\text{rd}(x) - x}{x} \right| \leq 2^{-t}$$

Jakie liczby, tzn. z jakiego zakresu, 'zna' komputer?

Niech

$$C_{\max} = 2^{d-t-1} - 1, \quad D := 2^{C_{\max}}.$$



W rzeczywistości w komputerze możemy reprezentować liczby ze zbioru $X' := (-D, D)$. W pamięci pojawiają się liczby ze zbioru dyskretnego $X_{fl} := \text{rd}(X')$.

Przykład. Rozważmy arytmetykę dla

$$d = 5, \quad t = 3.$$

Wtedy:

$$C_{\max} = 2^{d-t-1} - 1 = 2^1 - 1 = 1, \quad D = 2^{C_{\max}} = 2.$$

Możliwe mantysy:

$$m \in \left\{ \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8} \right\},$$

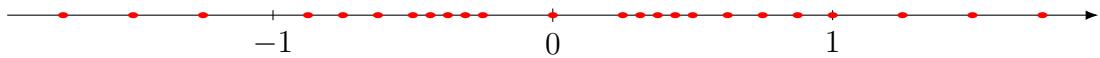
a cecha:

$$c \in \{-1, 0, 1\}.$$

Dodatnie liczby znormalizowane:

$$\left\{ \frac{1}{4}, \frac{5}{16}, \frac{3}{8}, \frac{7}{16}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1, \frac{5}{4}, \frac{3}{2}, \frac{7}{4} \right\}.$$

Zbiór X_{fl} jest symetryczny względem zera (dochodzi też liczba 0).



Zmiennopozycyjna realizacja działań arytmetycznych:

$$\text{fl}(x \circ y) := (x \circ y) (1 + \varepsilon_{x \circ y}), \quad \circ \in \{+, -, *, /\}, \quad x, y \in X_{fl}.$$

Błąd względny pojedynczej operacji spełnia:

$$|\varepsilon_{x \circ y}| \leq 2^{-t}.$$

Równoważnie:

$$\left| \frac{x \circ y - \text{fl}(x \circ y)}{x \circ y} \right| = |\varepsilon_{x \circ y}|.$$

Aby analizować (symulować) działanie algorytmów w arytmetyce zmiennopozycyjnej, będziemy często posługiwać się tzw. twierdzeniem o kumulacji błędów.

TW. (o kumulacji błędów)

Niech zachodzi

$$|\delta_i| \leq 2^{-t}, \quad i = 1, 2, 3, \dots, n,$$

oraz niech

$$1 + \sigma_n := \prod_{i=1}^n (1 + \delta_i).$$

Wtedy

$$\sigma_n = \sum_{i=1}^n \delta_i + O(2^{-2t}).$$

Jeśli dodatkowo

$$n \cdot 2^{-t} < 2,$$

to

$$|\sigma_n| \leq \gamma_n := \frac{n \cdot 2^{-t}}{1 - \frac{1}{2}n \cdot 2^{-t}} \approx n \cdot 2^{-t}.$$

Użyjemy twierdzenia o kumulacji błędów do analizy zmiennopozycyjnej realizacji prostego programu komputerowego.

Przykład. Rozważmy zadanie obliczenia sumy liczb x_1, x_2, x_3, x_4, x_5 , tzn.

$$S = \sum_{i=1}^5 x_i.$$

Wartość S wyznaczamy przy pomocy następującego ('naturalnego') programu:

```

 $S := x_1$ 
for  $i = 2$  to 5
     $S := S + x_i$ 
return  $S$ 
```

Jak wygląda zmiennopozycyjna realizacja programu?

Dla uproszczenia przyjmijmy, że

$$\text{rd}(x_i) = x_i, \quad 1 \leq i \leq 5$$

tj. dane wejściowe są liczbami maszynowymi.

Wtedy realizacja zmiennopozycyjna programu ma postać:

$$\text{fl}(P) = \left(\left(((x_1 + x_2)(1 + \xi_2) + x_3)(1 + \xi_3) + x_4 \right)(1 + \xi_4) + x_5 \right)(1 + \xi_5),$$

gdzie

$$|\xi_1|, |\xi_2|, |\xi_3|, |\xi_4|, |\xi_5| \leq 2^{-t}.$$

Po uporządkowaniu względem x_i :

$$\begin{aligned}
\text{fl}(P) = & x_1(1 + \xi_2)(1 + \xi_3)(1 + \xi_4)(1 + \xi_5) \\
& + x_2(1 + \xi_2)(1 + \xi_3)(1 + \xi_4)(1 + \xi_5) \\
& + x_3(1 + \xi_3)(1 + \xi_4)(1 + \xi_5) \\
& + x_4(1 + \xi_4)(1 + \xi_5) + x_5(1 + \xi_5).
\end{aligned}$$

Oznaczmy:

$$1 + E_i := \prod_{j=i}^5 (1 + \xi_j), \quad i = 2, 3, 4, 5, \quad E_1 := E_2.$$

Wówczas

$$\text{fl}(P) = \sum_{i=1}^5 x_i(1 + E_i).$$

Z twierdzenia o kumulacji błędów otrzymujemy (w pierwszym rzędzie):

$$|E_2| \leq \gamma_4 \lesssim 4 \cdot 2^{-t}, \quad |E_3| \leq \gamma_3 \lesssim 3 \cdot 2^{-t},$$

$$|E_4| \leq \gamma_2 \lesssim 2 \cdot 2^{-t}, \quad |E_5| = |\xi_5| \leq 2^{-t}.$$

Badamy błąd względny:

$$\left| \frac{S - \text{fl}(P)}{S} \right| = \left| \frac{\sum_{i=1}^5 x_i - \sum_{i=1}^5 x_i(1 + E_i)}{\sum_{i=1}^5 x_i} \right| = \left| \frac{\sum_{i=1}^5 x_i E_i}{\sum_{i=1}^5 x_i} \right|.$$

Stąd

$$\left| \frac{S - \text{fl}(P)}{S} \right| \leq \frac{\sum_{i=1}^5 |x_i| |E_i|}{\left| \sum_{i=1}^5 x_i \right|} \leq \left(\frac{\sum_{i=1}^5 |x_i|}{\left| \sum_{i=1}^5 x_i \right|} \right) \cdot 4 \cdot 2^{-t}.$$

Wprowadzamy oznaczenie:

$$K := \frac{\sum_{i=1}^5 |x_i|}{\left| \sum_{i=1}^5 x_i \right|}.$$

Wtedy

$$\left| \frac{S - \text{fl}(P)}{S} \right| \lesssim K \cdot 4 \cdot 2^{-t}.$$

Wniosek.

- Jeśli sumujemy liczby dodatnie, to warto je najpierw posortować.
- Jeśli wszystkie x_i mają ten sam znak, to $K = 1$.
- Może jednak być tak, że K jest dowolnie duże.

2.3 Zjawisko utraty cyfr znaczących

Problem utraty cyfr znaczących prześledźmy na przykładzie.

Niech

$$x, y \in X_{fl}, \quad x > y > 0, \quad x \approx y.$$

Przy odejmowaniu $x - y$ najpierw wyrównujemy cechy (przesuwamy mantysę jednej z liczb), a następnie odejmujemy mantisy. Gdy liczby są bliskie, najstarsze cyfry (bity) mantisy się redukują i wynik zaczyna się od wielu zer.

W efekcie:

- w wyniku zostaje mało cyfr znaczących,
- wzajemny błąd wyniku może istotnie wzrosnąć.

$$\begin{aligned} x &= + \boxed{1 \mid a_1 \mid a_2 \mid a_3 \mid a_4 \mid a_5} \cdot 2^c \\ &\quad \downarrow \text{wyrownanie cech} \\ y &= + \boxed{1 \mid a_1 \mid a_2 \mid a_3 \mid 0 \mid 0} \cdot 2^c \end{aligned}$$

$$\begin{aligned}
 x - y &= + \boxed{0 \mid 0 \mid 0 \mid b_1 \mid b_2 \mid b_3} \cdot 2^c \\
 &\quad \downarrow \text{normalizacja mantysy} \\
 &= + \boxed{1 \mid b_2 \mid b_3} \boxed{0 \mid 0 \mid 0} \cdot 2^{c-3}
 \end{aligned}$$

nie wiemy co tu wpisac

Przykład numeryczny (kod demonstracyjny).

```

f1:=z->ln(z)-1;

x:=exp(1.0001):

printf("      x = %1.10e\n\n",x);
printf("      f1(x) = %1.10e dla Digits:=%d\n ",f1(x),Digits);
printf(" Wynik dokładny = %1.30e\n\n",evalf(f1(x),64));

x = 2.7185536700e+00
f1(x) = 1.0000000000e-04 dla Digits:=10
Wynik dokładny = 9.999991401555060459381913048956e-05

f2:=z->ln(z/exp(1.0));

```

```

x:=exp(1.0001):

printf("      x = %1.10e\n\n",x);
printf("      f2(x) = %1.10e dla Digits:=%d\n ",f2(x),Digits);
printf(" Wynik dokładny = %1.30e\n\n",evalf(f1(x),64));

x = 2.7185536700e+00
f2(x) = 9.999999830e-05 dla Digits:=10
Wynik dokładny = 9.999991401555060459381913048956e-05

```