Andrzej Marciniak

ELEMENTY ANALIZY NUMERYCZNEJ

Wykłady dla studentów kierunku *informatyka*
Politechniki Poznańskiej

I. PODSTAWOWE POJĘCIA ANALIZY NUMERYCZNEJ

1.1. Stało- i zmiennopozycyjne przedstawienie liczby

Liczby są reprezentowane w komputerze przez skończoną liczbę cyfr ich rozwinięć pozycyjnych (podstawami są najczęściej 2, 8 lub 16). Wyróżniamy dwa sposoby reprezentacji liczb:

- stałopozycyjny,
- zmiennopozycyjny.

W reprezentacji stałopozycyjnej dowolną liczbę całkowitą l można przedstawić w postaci rozwinięcia podstawy. Zakładając, że podstawą jest liczba 2 mamy

$$l = z\sum_{i=0}^{n} e_i 2^i, \quad e_n \neq 0 \text{ dla } l \neq 0,$$

gdzie z oznacza znak liczby (+1 lub −1), a $e_i = 0$ lub 1. Jeśli dla reprezentacji stałopozycyjnej przeznaczono w komputerze $d + 1$ bitów, to liczbę l będzie można przedstawić tylko wówczas, gdy $n < d$.

Przykład 1.1

W 32-bitowej implementacji języka Delphi Pascal dla liczb typu ***Integer*** przeznaczono cztery bajty, czyli 32 bity. Jeśli jeden bit przeznaczono na znak liczby, to największą liczbą, którą można przedstawić na pozostałych 31 bitach jest ($n = 30$)

$$2^{30} + 2^{29} + \ldots + 2^1 + 2^0 = 2\,147\,483\,647.$$

Najmniejszą liczbą nie jest jednak (w zapisie dziesiętnym)

$$-2\,147\,483\,647,$$

gdyż liczby ujemne są zapisywane w tzw. *kodzie uzupełnieniowym*. W kodzie tym liczby ujemne są przedstawiane w postaci cyfr reprezentujących liczby jakie otrzymuje się odejmując wartości bezwzględne danych liczb od 2. Dlatego najmniejszą liczbą typu ***Integer*** jest

$$-2\,147\,483\,648,$$

która w komputerze na czterech bajtach jest zapisana następująco:

10000000 00000000 00000000 00000000.

Aby z zapisu liczb ujemnych w kodzie uzupełnieniowym otrzymać liczbę w zwykłym zapisie, należy na wszystkich bitach poza pierwszym zamienić cyfry 0 na 1, cyfry 1 na 0, a następnie dodać jedynkę dwójkową. ■

W zapisie stałopozycyjnym mogą być reprezentowane nie tylko liczby całkowite. Na przykład w języku Delphi Pascal w postaci takiej są pamiętane liczby rzeczywiste typu ***Currency***. Wewnętrzny zapis jest taki, jak liczb typu ***Integer***, tyle że na ośmiu bajtach, a nie na czterech, ale cztery najmniej znaczące cyfry są interpretowane jako cyfry po kropce dziesiętnej.

W przedstawieniu zmiennopozycyjnym, zwanym *znormalizowanym przedstawieniem półlogarytmicznym*, dowolną liczbę $x \neq 0$ przedstawia się w postaci

$$x = z2^c m,$$

gdzie z oznacza znak liczby, c – cechę (jest to liczba całkowita), a m oznacza mantysę, która jest liczbą rzeczywistą należącą do przedziału

$$\left[\frac{1}{2}, 1\right).$$

Liczba zero jest na ogół reprezentowane słowem o wszystkich bitach równych 0.

Załóżmy, że słowo do zapisu liczby rzeczywistej w przedstawieniu zmiennopozycyjnym ma długość $d + 1$ bitów. Jeśli t bitów przeznaczy się do zapisu mantysy, to na zapis cechy w sposób stałopozycyjny pozostanie $d - t$ bitów (jeden bit jest przeznaczony do zapisu znaku liczby). Przy takim podziale zamiast

$$m = \sum_{i=1}^{\infty} e_{-i} 2^{-i},$$

gdzie $e_{-1} = 1$ oraz $e_{-i} = 0$ lub 1 dla $i > 1$, zapamiętywanych jest w komputerze tylko t początkowych cyfr dwójkowych mantysy. Zakładając, że mantysa została prawidłowo zaokrąglona do t cyfr mamy

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

i stąd

$$\left| m - m_t \right| \leq \frac{1}{2} \cdot 2^{-t}.$$

Reprezentację zmiennopozycyjną liczby x oznaczamy przez $rd(x)$. Z definicji

$$rd(x) = z2^c m_t.$$

Mamy przy tym

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

co można też zapisać następująco:

$$rd(x) = x(1 + \varepsilon), \text{ gdzie } \left| \varepsilon \right| \leq 2^{-t}. \tag{1.1}$$

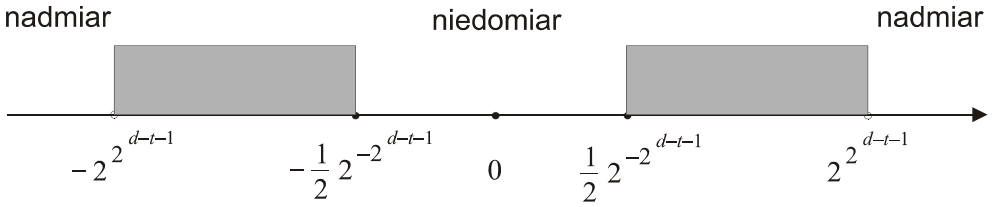
Liczbę $eps = 2^{-t}$ nazywa się *dokładnością maszynową*.

Liczba cyfr mantysy decyduje o dokładności zmiennopozycyjnego przedstawienia liczb, a liczba cyfr cechy określa zakres reprezentowanych liczb. Zakładając, że cecha jest zapisywana w zwykłym kodzie dwójkowym (nie w kodzie uzupełnieniowym) mamy

$$c \in \left[-2^{d-t-1}, 2^{d-t-1} \right] \text{ i } m_t \in \left[\frac{1}{2}, 1 \right),$$

a więc w komputerze możemy przedstawić w zapisie zmiennopozycyjnym liczbę 0 oraz liczby $x \neq 0$, dla których

$$\frac{1}{2} \cdot 2^{-2^{d-t-1}} \leq |x| < 2^{2^{d-t-1}}.$$



Rys. 1. Zakres liczb rzeczywistych reprezentowanych w komputerze

Zakres liczb w okolicy zera, ale bez zera, które nie mogą być reprezentowane w komputerze nazywa się *niedomiarem*, a zakresy liczb z przedziałów

$$\left(-\infty, -2^{2^{d-t-1}} \right] \text{ i } \left[2^{2^{d-t-1}}, +\infty \right)$$

nazywa się *nadmiarem*. Jeśli wynik operacji arytmetycznej wykonywanej w komputerze „wpadnie" do niedomiaru, to na ogół standardowo przyjmuje się, że jest on równy zero i ewentualne dalsze obliczenia są kontynuowane. Gdy wynik taki „wpadnie" do nadmiaru, to obliczenia są przerywane i następuje sygnalizacja błędu przekroczenia zakresu reprezentowanych liczb. W pewnych sytuacjach pierwszy przypadek może doprowadzić do niepoprawnych interpretacji. Przykładem może być rozwiązywanie układu równań liniowych, o którego wyznaczniku z teoretycznych rozważań może być wiadomo, że jest różny od 0, a więc którego rozwiązanie istnieje. Jeśli jednak wyznacznik ten będzie na tyle mały co do wartości bezwzględnej, że jego wartość (obliczona na komputerze) „wpadnie" do niedomiaru, to opierając się wyłącznie na obliczeniach komputerowych można dojść do wniosku, że rozwiązanie układu równań liniowych nie istnieje.

W większości współczesnych języków programowania w reprezentacji zmiennopozycyjnej liczb rzeczywistych cecha nie jest zapisywana ani w zwykłym kodzie binarnym, ani w kodzie uzupełnieniowym, lecz jest nieujemną liczbą całkowitą, którą interpretuje się odejmując od niej odpowiednią wartość wynikającą z liczby bitów przeznaczonych na cechę. Zyskuje się w ten sposób jeden bit (nie ma bitu przeznaczonego na znak cechy), co umożliwia interpretowanie pewnych zapisów jako dodatniej i ujemnej nieskończoności (*Inf*) oraz tzw. *nie-liczb* (*NaN*).

Przykład 1.2

W języku Delphi Pascal do zapisu liczb rzeczywistych w typie ***Single*** przeznaczono cztery bajty, z czego jeden bit na znak liczby, 8 bitów (jeden bajt) na cechę i 23 bity na mantysę. Wartość

cechy zapisanej na 8 bitach należy do przedziału [0, 255]. Jeśli oznaczymy przez s znak liczby ($s = 0$ lub 1), przez c – cechę, a przez m – mantysę, to wartość w liczby zapisanej w taki sposób jest określona następująco:

$$w = \begin{cases} (-1)^s 2^{(c-127)}(1.m), \text{ gdy } 0 < c < 255, \\ (-1)^s 2^{-126}(0.m), \text{ gdy } c = 0 \text{ i } m \neq 0, \\ (-1)^s 0, \text{ gdy } c = 0 \text{ i } m = 0, \\ (-1)^s Inf, \text{ gdy } c = 255 \text{ i } m = 0, \\ NaN, \text{ gdy } c = 255 \text{ i } m \neq 0. \end{cases}$$

■

Oznaczmy przez A zbiór liczb maszynowych, tj. zbiór liczb rzeczywistych, które mają dokładną reprezentację w komputerze. Wynik operacji na liczbach ze zbioru A nie musi być liczbą maszynową. Zwykle w komputerze zamiast matematycznych operacji dodawania (+), odejmowania (−), mnożenia (·) i dzielenia (/) dla liczb $x, y \in A$ realizowane są operacje $\oplus$, $\ominus$, $\odot$ i $\oslash$. Operacja $\oplus$ jest zdefiniowana następująco (podobnie są zdefiniowane operacje $\ominus$, $\odot$ i $\oslash$):

$$x \oplus y = rd(x + y), \quad x, y \in A,$$

skąd po uwzględnieniu (1.1) mamy

$$x \oplus y = (x + y)(1 + \tilde{\varepsilon}), \text{ gdzie } |\tilde{\varepsilon}| \leq eps. \tag{1.2}$$

Zauważmy, że dla operacji zmiennopozycyjnych nie obowiązują reguły operacji arytmetycznych. Na przykład, dla $x, y \in A$ jeśli

$$|y| < \frac{eps}{2}|x|,$$

to

$$x \oplus y = x.$$

Wartość wyrażenia α obliczoną w arytmetyce zmiennopozycyjnej przyjęto oznaczać przez $fl(\alpha)$. Jest oczywiste, że dla $x, y \in A$ mamy

$$fl(x + y) = rd(x + y),$$

ale jeśli $x, y \notin A$, to

$$fl(x + y) \neq rd(x + y).$$

Na podstawie (1.1) i (1.2) mamy bowiem

$$\begin{aligned} fl(x + y) &= rd(x) \oplus rd(y) = [rd(x) + rd(y)](1 + \tilde{\varepsilon}) \\ &= [x(1 + \varepsilon_1) + y(1 + \varepsilon_2)](1 + \tilde{\varepsilon}) = (x + y)\left(1 + \frac{x\varepsilon_1 + y\varepsilon_2}{x + y}\right)(1 + \tilde{\varepsilon}), \end{aligned}$$

gdzie

$$|\varepsilon_1|, |\varepsilon_2|, |\tilde{\varepsilon}| \leq eps.$$

Dla operacji zmiennopozycyjnych nie zachodzą prawa łączności i rozdzielności. Może okazać się, że dla liczb rzeczywistych a, b i c mamy

$$fl((a + b) + c) \neq fl(a + (b + c)).$$

Ostatnią zależność można uogólnić: dwie różne, ale matematycznie równoważne metody obliczania wartości pewnego wyrażenia w rachunku zmiennopozycyjnym mogą dać różne wyniki.

1.2. Uwarunkowanie zadania, numeryczna poprawność i stabilność

Rozważmy zadanie polegające na obliczeniu dla danych

$$x = (x_1, x_2, \dots, x_n)$$

wyniku

$$y = (y_1, y_2, \dots, y_m),$$

tj.

$$y = \varphi(x), \quad \varphi\colon \mathbf{R}^n \supset D \to \mathbf{R}^m,$$

przy czym odwzorowanie φ jest ciągiem odwzorowań (algorytmem):

$$\varphi = \varphi^{(r)} \circ \varphi^{(r-1)} \circ \dots \circ \varphi^{(0)}, \text{ gdzie } \varphi^{(i)}\colon D_i \to D_{i+1},\ i = 0, 1, \dots, r,$$
$$D_j \subseteq \mathbf{R}^{n_j}, \quad D_0 = D, \quad D_{r+1} \subseteq \mathbf{R}^m.$$

Przykład 1.3

Niech $\varphi(x_1, x_2, x_3) = x_1 + x_2 + x_3$. Rozważmy dwa algorytmy:

1) $\eta = x_1 + x_2,\ y = \eta + x_3,$
2) $\eta = x_2 + x_3,\ y = x_1 + \eta,$

czyli dla pierwszego algorytmu mamy

$$\varphi^{(0)}(x_1, x_2, x_3) = \begin{bmatrix} x_1 + x_2 \\ x_3 \end{bmatrix} \in \mathbf{R}^2, \quad \varphi^{(1)}(u_1, u_2) = u_1 + u_2 \in \mathbf{R},$$

a dla drugiego algorytmu jest

$$\varphi^{(0)}(x_1, x_2, x_3) = \begin{bmatrix} x_1 \\ x_2 + x_3 \end{bmatrix} \in \mathbf{R}^2, \quad \varphi^{(1)}(u_1, u_2) = u_1 + u_2 \in \mathbf{R}.$$

Przy wykonywaniu obliczeń w rachunku zmiennopozycyjnym zamiast y otrzymujemy

$$\bar{y} = fl((x_1 + x_2) + x_3) \text{ lub } \bar{\bar{y}} = fl(x_1 + (x_2 + x_3)).$$

Dla pierwszego algorytmu otrzymujemy

$$\bar{\eta} = fl(x_1 + x_2) = (x_1 + x_2)(1 + \varepsilon_1),$$
$$\bar{y} = fl(\bar{\eta} + x_3) = (\bar{\eta} + x_3)(1 + \varepsilon_2) = [(x_1 + x_2)(1 + \varepsilon_1) + x_3](1 + \varepsilon_2)$$
$$= (x_1 + x_2 + x_3)\left[1 + \frac{x_1 + x_2}{x_1 + x_2 + x_3}\varepsilon_1(1 + \varepsilon_2) + \varepsilon_2\right].$$

Obliczmy błąd względny wyniku. Mamy

$$\varepsilon_{\bar{y}} = \frac{\bar{y} - y}{y} = \frac{x_1 + x_2}{x_1 + x_2 + x_3}\varepsilon_1(1 + \varepsilon_2) + \varepsilon_2 \doteq \frac{x_1 + x_2}{x_1 + x_2 + x_3}\varepsilon_1 + 1 \cdot \varepsilon_2,$$

gdzie symbol $\doteq$ oznacza „w pierwszym przybliżeniu". W drugim algorytmie dostaniemy

$$\varepsilon_{\bar{\bar{y}}} \doteq \frac{x_2 + x_3}{x_1 + x_2 + x_3}\varepsilon_3 + 1 \cdot \varepsilon_4,$$

gdzie $|\varepsilon_i| \le eps$ $(i = 1, 2, 3, 4)$. Współczynniki przy ε_i nazywają się *współczynnikami wzmocnienia* i określają wpływ błędów zaokrągleń na błąd wyniku. Oczywiście lepszy jest ten algorytm, dla którego współczynniki te są mniejsze. W przypadku dodawania w arytmetyce zmiennopozycyjnej trzech liczb oznacza to, że najpierw należy dodać dwie mniejsze liczby. ■

Powyższy przykład pokazuje w jaki sposób błędy zaokrągleń wpływają na błąd wyniku. Należy jeszcze zbadać wpływ błędów danych wejściowych. Załóżmy zatem, że zamiast dokładnych wartości x_i mamy wartości przybliżone $\tilde{x}_i$. Oznaczmy przez

$$\Delta x_i = \tilde{x}_i - x_i$$

błąd bezwzględny wartości x_i, a przez

$$\varepsilon_{x_i} = \frac{\Delta x_i}{x_i} \quad (x_i \neq 0)$$

błąd względny tych wartości.

Stosując algorytm φ otrzymujemy zamiast wartości $y = \varphi(x)$ wartość $\tilde{y} = \varphi(\tilde{x})$. Rozważmy błąd bezwzględny wartości y_i $(i = 1, 2, \ldots, m.)$. Korzystając ze wzoru Taylora dla funkcji wielu zmiennych mamy

$$\Delta y_i = \tilde{y}_i - y_i = \varphi_i(\tilde{x}) - \varphi_i(x) \doteq \sum_{j=1}^{n} (\tilde{x}_j - x_j)\frac{\partial \varphi_i(x)}{\partial x_j} = \sum_{j=1}^{n} \frac{\partial \varphi_i(x)}{\partial x_j}\Delta x_j.$$

Stąd

$$\varepsilon_{y_i} \doteq \sum_{j=1}^{n} \frac{x_j}{\varphi_i(x)}\frac{\partial \varphi_i(x)}{\partial x_j}\varepsilon_{x_j}. \tag{1.3}$$

Definicja 1.1. Wielkości charakteryzujące wpływ zaburzeń danych na zaburzenia rozwiązania nazywamy *wskaźnikami uwarunkowania zadania*. Jeśli wielkości te są co do wartości bezwzględnej duże, to mówimy, że *zadanie jest źle uwarunkowane*.

Dla algorytmu $y = \varphi(x)$ wskaźnikami uwarunkowania są wielkości

$$\frac{x_j}{\varphi_i(x)}\frac{\partial \varphi_i(x)}{\partial x_j}, \quad i = 1, 2, \ldots, m, \; j = 1, 2, \ldots, n.$$

Jest to $m \cdot n$ liczb. Gdy wielkości m i n są duże, posługiwanie się tak wielką liczbą współczynników uwarunkowania staje się uciążliwe. Dlatego często korzysta się z prostszej definicji wskaźnika uwarunkowania, który jest pojedynczą liczbą.

Definicja 1.2. *Wskaźnikiem uwarunkowania zadania* nazywamy liczbę c, dla której

$$\frac{\left\| \varphi(\tilde{x}) - \varphi(x) \right\|}{\left\| \varphi(x) \right\|} \le c \frac{\left\| \tilde{x} - x \right\|}{\left\| x \right\|}, \quad x, \varphi(x) \neq 0.$$

Symbol $\| . \|$ oznacza normę[1].

Z wzoru (1.3) mamy

$$\varphi(x_1, x_2) = x_1 \cdot x_2 \quad \Rightarrow \quad \varepsilon_{x_1 \cdot x_2} \doteq \varepsilon_{x_1} + \varepsilon_{x_2},$$

$$\varphi(x_1, x_2) = x_1 / x_2 \quad \Rightarrow \quad \varepsilon_{x_1 / x_2} \doteq \varepsilon_{x_1} - \varepsilon_{x_2},$$

$$\varphi(x_1, x_2) = x_1 \pm x_2 \quad \Rightarrow \quad \varepsilon_{x_1 \pm x_2} \doteq \frac{x_1}{x_1 \pm x_2} \varepsilon_{x_1} \pm \frac{x_2}{x_1 \pm x_2} \varepsilon_{x_2}, \quad x_1 \pm x_2 \neq 0.$$

Wynika stąd, że przy mnożeniu i dzieleniu względne błędy danych słabo przenoszą się na wynik. Przy dodawaniu lub odejmowaniu małe błędy względne danych mogą powodować duży błąd względny wyniku.

Przykład 1.4

Niech

$$x_1 = 1{,}0005, \ \tilde{x}_1 = 1{,}001, \ x_2 = -1{,}0004, \ \tilde{x}_2 = -1{,}000.$$

Mamy

$$\varepsilon_{x_1} = 5 \cdot 10^{-4}, \ \varepsilon_{x_2} = -4 \cdot 10^{-4},$$

a więc błędy względne danych są małe. Błąd względny przy dodawaniu będzie jednak duży. Mamy bowiem

$$\varepsilon_{x_1 + x_2} \doteq \frac{1{,}0005}{10^{-4}} 5 \cdot 10^{-4} + \frac{-1{,}0004}{10^{-4}} (-4 \cdot 10^{-4}) > 9. \qquad \blacksquare$$

Jak już wspomniano, zadanie numeryczne jest problemem polegającym na wyznaczeniu wektora wyników y na podstawie wektora danych x przy zastosowaniu pewnego algorytmu φ.

Definicja 1.3. Mówimy, że *zadanie jest dobrze postawione*, jeżeli wektor y jest jednoznacznie określony dla przyjętego wektora danych x.

Niech $\bar{y}$ oznacza obliczony numerycznie wektor wyników y.

Definicja 1.4. Algorytm obliczania $\bar{y}$ nazywamy *poprawnie sformułowanym*, gdy liczba niezbędnych działań nad wektorem danych x jest skończona (choć może zależeć od x).

[1] Zob. definicję normy w dowolnym podręczniku z analizy matematycznej lub analizy funkcjonalnej.

Przykład 1.5

Niech będzie dana liczba zespolona $z = a + ib$, gdzie i oznacza jednostkę urojoną. Należy obliczyć $\frac{1}{z^2}$. Jednym z algorytmów rozwiązujących to zadanie może być następujący:

- oblicz $t = \frac{b}{a}$ (tangens fazy liczby z),
- oblicz $|z|^2 = a^2 + b^2$ (kwadrat modułu liczby z),
- wyznacz $\mathrm{Re}\left(\frac{1}{z^2}\right) = \frac{1}{|z|^2}\frac{1-t^2}{1+t^2}$ oraz $\mathrm{Im}\left(\frac{1}{z^2}\right) = \frac{1}{|z|^2}\frac{-2t}{1+t^2}$.

Powyższe zadanie jest dobrze postawione, jeśli $a^2 + b^2 \neq 0$. Oznacza to, że dziedziną rozwiązania zadania jest $D = \mathbf{R}^2 - \{(0, 0)\}$. Algorytm jest poprawnie sformułowany (można sprawdzić, że jest w nim 11 niezbędnych działań), ale nie dla każdej pary $(a, b) \neq (0, 0)$ można tym algorytmem znaleźć rozwiązanie. Podczas wykonywania obliczeń na komputerze może wystąpić błąd dzielenia przez zero i błąd nadmiaru. Jest oczywiste, że pierwszy błąd wystąpi, gdy $a = 0$, ale błąd dzielenia przez zero wystąpi także wówczas, gdy liczba a będzie wprawdzie różna od zera, ale tak mała, że w komputerze będzie reprezentowana przez zero („wpadnie" do niedomiaru). Ponadto, jeśli największa liczba, którą można reprezentować w komputerze będzie rzędu np. 10^{80}, to dla liczby b rzędu 10^{50} i liczby a rzędu 10^{-50} wystąpi nadmiar (z uwagi na dzielenie b przez a). Oznacza to, że na komputerze zadanie może być rozwiązane w dziedzinie $\overline{D} \subset D$, ale niekoniecznie $\overline{D} = D$. Zbiór $\overline{D}$ zależy przy tym od właściwości komputera, a dokładniej od reprezentacji w nim liczb rzeczywistych. ■

Definicja 1.5. Mówimy, że *algorytm φ jest numerycznie stabilny*, jeśli dla dowolnie wybranych danych $x_0 \in D$ istnieje taka dokładność obliczeń δ_0, że dla dokładności $\delta < \delta_0$ mamy $x_0 \in \overline{D}(\delta)$ oraz

$$\lim_{\delta \to 0} \overline{\varphi}(x_0, \delta) = \varphi(x_0),$$

gdzie $\overline{\varphi}$ oznacza algorytm φ zależny od rodzaju arytmetyki komputera.

Innymi słowy powyższa definicja mówi, że algorytm jest numerycznie stabilny wtedy, gdy zwiększając dokładność obliczeń można wyznaczyć (z dowolną dokładnością) dowolne istniejące rozwiązanie zadania.

Przykład 1.6

Algorytm z poprzedniego przykładu, choć poprawnie sformułowany, nie jest numerycznie stabilny, gdyż dla $a = 0$ (bez względu na wartość b) nie można nim wyznaczyć rozwiązania zadania przez wzrost dokładności obliczeń. Inny algorytm wyznaczenia $\frac{1}{z^2}$ może być następujący:

- oblicz $\mathrm{Re}\left(\frac{1}{z^2}\right) = \frac{a^2 - b^2}{(a^2 + b^2)^2}$ oraz $\mathrm{Im}\left(\frac{1}{z^2}\right) = \frac{-2ab}{(a^2 + b^2)^2}$,

który jest poprawnie sformułowany (występuje w nim 9 działań) i jest przy tym numerycznie stabilny. Stabilność wynika z ciągłości podanych wzorów przy założeniu $a^2 + b^2 \neq 0$. ■

Dla wielu zadań znane są różne metody i algorytmy ich rozwiązywania. Spośród różnych algorytmów chcielibyśmy zawsze wybrać najlepszy. Jednym z kryteriów wyboru może być jakość otrzymanych rozwiązań. Innym ważnym czynnikiem może być koszt danej metody mierzony liczbą obliczeń, przy czym jeśli porównamy tylko koszt znanych metod, to problemu nie rozstrzygniemy. Znajomość i zastosowanie najlepszej spośród znanych metod nie przeczy istnieniu jeszcze lepszej metody.

Stosunkowo niedawno zaczęto poszukiwać i badać metody optymalne. Przyjmując za kryterium liczbę działań arytmetycznych potrzebną do rozwiązania zadania, metodą optymalną będzie metoda minimalizująca tę liczbę. Problemami istnienia i własności metod optymalnych oraz ich konstrukcji zajmuje się dział analizy numerycznej zwany złożonością obliczeniową. Okazuje się, że wiele klasycznych metod nie spełnia postulatu optymalności i tylko dla niewielu prostych zadań znamy metody optymalne.

Rozważmy ponownie zadanie obliczenia

$$y = \varphi(x), \quad \varphi\colon \mathbf{R}^n \supset D \to \mathbf{R}^m.$$

Definicja 1.6. Wielkość

$$z(\varphi, D) = \sup_{x \in D} z(\varphi, x),$$

gdzie $z(\varphi, x)$ oznacza minimalną liczbę działań potrzebną do obliczenia $\varphi(x)$, nazywamy *złożonością obliczeniową zadania* $y = \varphi(x)$.

Definicja 1.7. Mówimy, że zadanie $y = \varphi(x)$ ma *n istotnych danych* na zbiorze D, jeśli istnieją dane $x = (x_1, x_2, \dots, x_n) \in D$, dla których zmiana dowolnej ze składowych x_j ($j = 1, 2, \dots, n$) powoduje zmianę wyniku, tj.

$$\exists_{x \in D} \; \forall_{j = 1, 2, \dots, n} \; \exists_{\alpha} \; x + \alpha e_j \in D \wedge \varphi(x) \neq \varphi(x + \alpha e_j),$$

gdzie $e_j = (0, \dots, 0, 1, 0, \dots, 0)^T$ (j-ty wektor jednostkowy).

W przypadku, gdy algorytm oprócz działań arytmetycznych obejmuje też działania logiczne ograniczamy zbiór D tak, aby wyniki tych działań nie zależały od x.

Udowodniono, że jeśli zadanie $y = \varphi(x)$ ma n istotnych danych, to minimalna liczba działań $z(\varphi, D) \geq n / 2$. Oznacza to, że liczba istotnych danych określa oszacowanie z dołu złożoności obliczeniowej. Oszacowanie to jest często realistyczne, gdyż dla wielu zadań o n istotnych danych znane są algorytmy, które wymagają wykonania Cn działań, gdzie C oznacza stałą rzędu jedności. Istnieją jednak zadania, dla których najtańsze ze znanych algorytmów rozwiązują je kosztem wielokrotnie większej liczby działań niż liczba danych.

Dalsze wiadomości na temat zasygnalizowanych tu zagadnień można znaleźć m. in. w podręcznikach [1] – [3], [5] i [6]. Na zakończenie tego rozdziału przytaczamy zestaw czynników, które należy brać pod uwagę przy numerycznym rozwiązywaniu dowolnego problemu. Możemy je podzielić na cztery grupy i spróbować odpowiedzieć na każde z podanych pytań.

- Dane
  - Jakie wielkości są danymi rozwiązywanego zadania?
  - Jakie przestrzenie danych i wyników (ich struktury i normy) najlepiej odpowiadają sensowi fizycznemu rozwiązywanego zadania?
- Uwarunkowanie zadania
  - Czy zadanie nie jest zbyt wrażliwe na zaburzenia danych?
  - Czy w danej arytmetyce zadanie w ogóle można rozwiązać?
  - Czy istnieje zadanie równoważne danemu, ale lepiej uwarunkowane?

  Przy odpowiedziach negatywnych na dwa ostatnie pytania należy stosować arytmetykę o wyższej precyzji.
- Jakość numeryczna stosowanego algorytmu
  - Czy algorytm jest numerycznie stabilny?
  - Czy algorytm jest numerycznie poprawny?
- Efektywność stosowanej metody
  - Jaka jest lub co wiemy o złożoności obliczeniowej zadania?
  - Jaka jest efektywność innych metod rozwiązujących dany problem?
  - Czy rozpatrywana metoda jest optymalna, a jeśli nie, czy jest to najtańsza ze znanych metod?

Zadania

1. Wykazać, że jeśli liczba x jest liczbą maszynową, to dla dowolnego naturalnego n jest

$$fl(x^n) = x^n(1+\varepsilon)^{n-1}, \text{ gdzie } |\varepsilon| < eps.$$

2. Różnicę kwadratów dwóch liczb można obliczyć z wzoru

$$a \cdot a - b \cdot b$$

lub

$$(a+b)(a-b).$$

Realizacja którego wzoru (algorytmu) w arytmetyce zmiennopozycyjnej jest lepsza i dlaczego?

3. Kiedy zadanie wyznaczenia $y = \varphi(x_1, x_2, x_3) = x_1 + x_2 + x_3$ jest dobrze uwarunkowane?

4. Dla jakich wartości danych zadanie obliczenia $y = \varphi(x_1, x_2) = -x_1 + \sqrt{x_1 + x_2}$ jest źle uwarunkowane?

5. Wskaźnik uwarunkowania obliczenia wartości $y = \varphi(x) = x^\alpha$ nie zależy od x. Jaki on jest?