Spis treści

Spis treści

Spis treści

Spis treści

Spis treści

Spis treści

Spis treści

Literatura

1. Z. Fortuna, B. Macukow, J. Wąsowski, Metody numeryczne, WNT, Warszawa, 1993
2. A. Ralston, Wstęp do analizy numerycznej, PWN, Warszawa, 1983
3. E. Kącki, A. Małolepszy, A. Romanowicz, Metody numeryczne dla inżynierów, Wyższa Szkoła Informatyki w Łodzi, Łódź, 2005
4. Z. Kosma, Metody numeryczne dla zastosowań inżynierskich, Politechnika Radomska, Radom, 2008
5. W.T. Vetterling, S.A. Teukolsky, W.H. Press, B.P. Flannery, Numerical Recipes, Cambridge University Press, 2003

Rodzaje błędów

Zadanie

Rozwiązać układ równań matematycznych (całkowych, różniczkowych, algebraicznych . . .), przy czym sama interpretacja otrzymanych wyników leży chwilowo poza zakresem naszych zainteresowań.

Podstawowe rodzaje błędów, jakie mogą pojawić się w procesie obliczeniowym:

- błędy danych wejściowych,
- błędy obcięcia,
- błędy zaokrągleń.

Błędy danych wejściowych

Większość obliczeń komputerowych wykonuje się na liczbach. Proces obliczeniowy można sobie wyobrazić jako czarną skrzynkę, do której wsypuje się dane wejściowe, a wysypuje się wyniki, czyli przemielone w tej skrzynce dane wejściowe. Naszym celem jest spowodowanie, by ta czarna skrzynka przybrała inny kolor, powiedzmy, szary.

Błędy danych wejściowych

Dane wejściowe dostarczone komputerowi mają różne źródła pochodzenia. Jeśli są one wynikiem pomiarów (np. wymiary geometryczne, czas, wartości wielkości fizycznych i.t.p.), to wiadomo, że pomiar nigdy nie będzie dokładny, zawsze jest obarczony jakimś błędem. Również i tu przyczyny błędów są dwojakie:

- wynikające z niedokładności przyrządów pomiarowych
- wynikające z natury rzeczy

Błędy danych wejściowych

Błędy wynikające z natury rzeczy

Zasada nieoznaczoności Heisenberga

1. dla położenia i pędu

$$\triangle x \cdot \triangle p \geq \hbar\,, \tag{1}$$

2. dla energii E i czasu t

$$\triangle E \cdot \triangle t \geq \hbar\,, \tag{2}$$

$\hbar$ – stała Plancka

Zasada ta stosuje się co prawda tylko do układów kwantowych, gdy np. chcemy znaleźć bardzo dokładne położenie jakiejś cząstki elementarnej (elektronu, protonu, mezonu...), tym nie mniej warto zawsze o niej pamiętać i ze skromnością podchodzić do uzyskanych wyników pomiarowych.

Błędy danych wejściowych

Nie zawsze zależy nam na bardzo dokładnych pomiarach.

Przykład:
zmierzyć odległość z dokładnością do milimetrów pomiędzy odległymi obiektami kosmicznymi! – nie bardzo widać sens w dążeniu do takiej dokładności.

Błędy danych wejściowych

W podobnych przypadkach w obliczeniach wartość dokładną liczby A zastępujemy jej wartością przybliżoną a. Rozróżniamy tu dwa przypadki:

- a jest wartością przybliżoną liczby A z nadmiarem, jeśli $a > A$;
- a jest wartością przybliżoną liczby A z niedomiarem, jeśli $a < A$;

Różnicę pomiędzy wartością liczby A a jej wartością przybliżoną a

$$\triangle A = A - a, \tag{3}$$

nazywamy **błędem liczby** A.

Błędy obcięcia

Często okazuje się, że dane zagadnienie możemy rozwiązać tylko w sposób przybliżony i wiemy to od samego początku. Powodem może być na przykład konieczność zastąpienia nieskończonego szeregu poprzez ograniczoną liczbę jego składników, co ma miejsce na przykład w przypadku obliczania całek oznaczonych lub rozwinięć w szereg wyrażeń arytmetycznych.

$$e^x = 1 + x + \frac{1}{2}x^2 + \ldots + \frac{1}{N!}x^N + \ldots \tag{4}$$

Pojawiający się wtedy błąd obliczeniowy nazywamy **błędem metody** lub **błędem obcięcia**.
Wielkość tego błędu można oszacować i będziemy starali się to robić przy okazji omawiania różnych algorytmów.

Błędy obcięcia

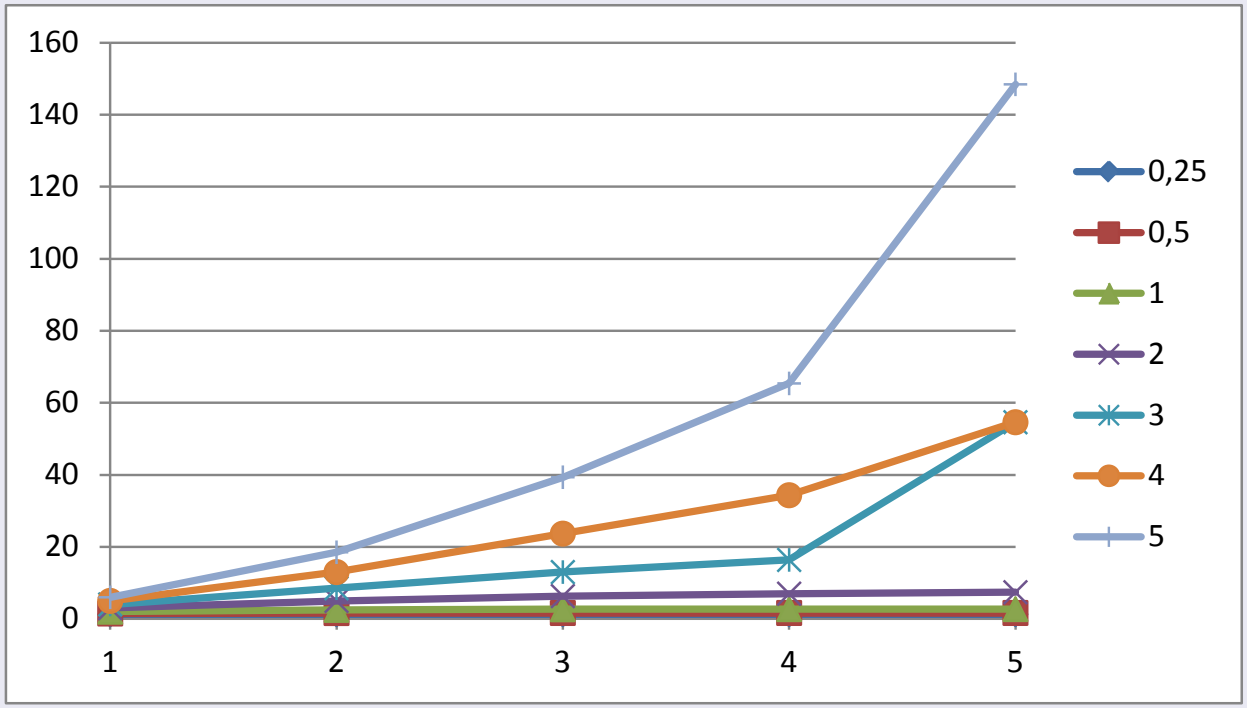
Przykład

x	1+x	x^2/2	x^3/6	x^4/24	Excel
0,25	1,25	1,28125	1,283854	1,284016927	1,284025
0,5	1,5	1,625	1,645833	1,6484375	1,648721
1	2	2,5	2,666667	2,708333333	2,718282
2	3	5	6,333333	7	7,389056
3	4	8,5	13	16,375	54,59815
4	5	13	23,66667	34,33333333	54,59815
5	6	18,5	39,33333	65,375	148,4132

Rys. 1: Tabelka z przybliżeniami funkcji $f(x) = e^x$

Błędy obcięcia

Przykład



Rys. 2: Wykresy z przybliżeniami funkcji $f(x) = e^x$

Błędy zaokrągleń

Mówiąc o błędach, trzeba zawsze precyzyjnie rozróżniać ich dwie podstawowe definicje. Tak więc mamy:

- błąd bezwzględny (absolutny)(ozn.: BA), gdzie
  wartość dokładna = wartość przybliżona + błąd absolutny
- błąd względny (ozn.: BW), gdzie

$$\text{błąd względny} = \frac{\text{błąd absolutny}}{\text{wartość dokładna}}$$

Błędy zaokrągleń

Przykład

Jeśli przybliżymy ułamek okresowy $1/3$ poprzez 0.333, to mamy

$$BA = \frac{1}{3} - \frac{333}{1000} = \frac{1}{3} \cdot 10^{-3};$$

$$BW = \frac{\frac{1}{3} \cdot 10^{-3}}{\frac{1}{3}} = 10^{-3}.$$

Błędy zaokrągleń

Bardzo często wartość dokładna jest nieznana i wtedy można skorzystać z definicji podanych, na przykład, przez Łukaszewicza i Warmusa

- błąd bezwzględny wartości przybliżonej to dowolna liczba nie mniejsza od wartości bezwzględnej (modułu) różnicy wartości dokładnej i wartości przybliżonej;

  błąd ten nabiera większego znaczenia w przypadku złożonych obliczeń numerycznych, a definicja błędu bezwzględnego podana powyżej jako pierwsza, określa w rzeczywistości minimalny błąd absolutny;

- błąd względny jest to iloraz błędu bezwzględnego i modułu wartości przybliżonej.

Podane tu przez nas definicje błędów są różne, dlatego podając wartości liczbowe błędów, warto określić, którą z tych definicji mamy na myśli.

Reprezentacja liczb w komputerze

Liczby w komputerze reprezentowane są poprzez ciągi bitów, pogrupowane w większe jednostki:

- bajty (bytes)
- słowa (words), czyli grupy bajtów

Wraz z rozwojem technologii komputerowej długość słowa uległa stopniowemu wydłużeniu i obecnie wynosi najczęściej 8 bajtów, czyli 64 bity.

Reprezentacja liczb w komputerze

Wielkości te wynikają z historycznego rozwoju komputerów i z kultury śródziemnomorskiej, co zaowocowało ustaleniem pewnych standardów w komputerologii, np. zestawem kodów ASCII[a]. Został on ustalony dosyć dawno temu (siedmiobitowy standard został zaproponowany przez ANSI[b] w roku 1963, a przyjęty w 1968), więc nie dziwi, że obecnie znaki niedrukowalne są dosyć rzadko używane.

[a] ASCII - American Standard Code for Information Interchange
[b] ANSI - American National Standards Institute

Kody ASCII

Jeśli przyjmiemy, że bit najbardziej znaczący, czyli mający numer 7, jest bitem znaku, czyli tzw. bitem parzystości, to pozostałe 7 bitów pozwala zakodować

$$1111111_2 = 127_{10}$$

127 znaków. Są to duże i małe litery alfabetu łacińskiego, cyfry, symbole operacji matematycznych i szereg znaków specjalnych.

Unicode

Oczywiście, takie były początki, kiedy komputery były wprowadzane tylko w obszarze języka anglo-amerykańskiego. Obecnie rozwój technologii pozwala na używanie w komputerach języków narodowych, co z kolei zaowocowało bałaganem, czyli pojawieniem się olbrzymiej liczby „standardów" w każdym języku, a następnie intensywnymi pracami nad ustanowieniem jednego standardu, który nazwano Unicode[a], gdzie każdy znak reprezentowany jest przez słowo czterobajtowe, czyli przez 64 bity. Zaletą tego nowego podejścia jest fizyczna możliwość reprezentowania wszystkich znaków narodowych, natomiast wadą olbrzymie straty niewykorzystanej pamięci.

[a] patrz http://www.unicode.org/unicode/standard/translations/polish.html

Reprezentowanie liczb w komputerach

Gdyby liczby były reprezentowane znakowo, czyli gdyby każda liczba składała się ze znaków cyfr, a każda cyfra była reprezentowana przez bajt pamięci, to łatwo zauważyć, że natychmiast pojawia się problem chociażby z takimi matematycznymi operacjami jak „ręczne” mnożenie liczb

$$\begin{array}{r} 12 \\ \times\ 71 \\ \hline 12 \\ 84 \\ \hline 852 \end{array} \tag{5}$$

Pozycyjny zapis liczb

Wartość przypisywana cyfrze w liczbie wyznacza nie tylko ona sama, ale i jej położenie w ciągu znaków definiujących liczbę:

$$\begin{aligned} 11111_2 &= 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 32 + 16 + 8 + 0 + 2 + 1 = 59_{10} \\ 59_{10} &= 5 \cdot 10^1 + 9 \cdot 10^0 \end{aligned} \tag{6}$$

Operacje logiczne

Wprowadzenie do komputerów reprezentacji zero-jedynkowej pozwala na ujednolicenie operacji arytmetycznych i logicznych. Na ogół fałsz logiczny (**FALSE**) jest reprezentowany przez 0 (zero), a wszystko co nie jest zerem, a więc na przykład jedynką, jest prawdą (**TRUE**).
Przypatrzmy się dwu tabelom, w których reprezentujemy arytmetyczne dodawanie w systemie dwójkowym i dodawanie (operacja logiczna „lub") wyrażeń logicznych Jeśli $F = 0$, a $T = 1$, to są one identyczne.

$+$	0	1
0	0	1
1	1	1

$\vee$	F	T
F	F	T
T	T	T

Operacje logiczne

Podobnie jest z mnożeniem i operacją logiczną "i".

$\cdot$	0	1
0	0	0
1	0	1

$\wedge$	F	T
F	F	F
T	F	T

Reprezentacja liczb rzeczywistych

Każda liczba w komputerze jest reprezentowana przez skończony ciąg bitów, a ten fakt nabiera naprawdę istotnego znaczenia gdy liczby są albo bardzo duże albo bardzo małe.
Reprezentowanie liczb w maszynie cyfrowej w tzw. *postaci zmiennopozycyjnej*: każda liczba zapisywana jest w następującej postaci

$$x = M \times N^W, \quad \frac{1}{2} \leq M < 1$$

gdzie
M – mantysa liczby x,
N – podstawa części potęgowej, na ogół 2 lub 10, ale w IBM 360 wynosi 16,
W – wykładnik części potęgowej.

Reprezentacja liczb rzeczywistych

Jeśli przyjmiemy, że w naszej maszynie wirtualnej w zapisie dwójkowym na M mamy do dyspozycji 5 bitów, a na W 3 bity, to reprezentacja

$$x = (1)1101\ (0)10$$

oznacza, że

$$\begin{aligned} x &= -0,1101 \times 2^{+10}\big|_2 \\ &= -\left(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}\right) \times 2^{+(1\cdot 2+0\cdot 1)}\Big|_{10} \\ &= -3,25|_{10} \end{aligned}$$

Przykład

Wskazaliśmy, że błąd reprezentacji zależy od wielkości liczby. W proponowanej reprezentacji można przedstawić liczby z zakresu $\{(-7.5,\ -0.0625), 0, (0.0625,\ 7.5)\}$.

Przykład 1

$$x = 0,2|_{10} = 0.0011(0011)\ldots|_2$$
$$x' = (0)1100\ (1)10 \equiv 0.001100|_2 = 0.1875|_{10}$$

Błąd bezwzględny: $|x - x'| = 0.0125$

Przykład 2

$$x = -6.25|_{10} = 0.0011(0011)\ldots|_2$$

$$x' = (1)1101\ (0)11 \equiv -110,1|_2 = -6.5|_{10}$$

Błąd bezwzględny: $|x - x'| = 0.25$
Jeśli zdefiniujemy błąd względny ε' jako taką liczbę, że

$$x' = x(1 + \varepsilon')$$

to dla 0.2 mamy $\varepsilon' = -0.0625$, a dla -6.26 mamy $\varepsilon' = 0.04$

Lemat Wilkinsona

Lemat Wilkinsona

Błędy zaokrągleń powstające podczas wykonywania działań zmiennopozycyjnych są równoważne pewnemu zaburzeniu liczb biorących udział w tych działaniach. I tak:

$\mathrm{fl}\ (x_1 \pm x_2) = x_1(1+\varepsilon_1) \pm x_2(1+\varepsilon_2)$

$\mathrm{fl}\ (x_1 \ x_2) = x_1(1+\varepsilon_3) \cdot x_2 = x_1 \cdot x_2(1+\varepsilon_3)$

$\mathrm{fl}\ (x_1/x_2) = x_1(1+\varepsilon_4)/x_2 = x_1/(x_2(1+\varepsilon_5))$

gdzie liczby $\varepsilon_i, \quad i = 1, \ldots, 5$, są nie większe co do modułu od pewnego parametru ε, charakteryzującego dokładność maszyny cyfrowej. Symbol *fl* oznacza, że działanie zostało wykonane w technice zmiennopozycyjnej.

Przykład zastosowania lematu Wilkinsona

Policzmy sumę dwu liczb:
$(0)1111\ (1)01 = 0.46875$ i $(0)1111\ (1)10 = 0.234375$
Dokładny wynik wynosi

$$(0)1111\ (1)01 + (0)1111\ (1)10 = 0.01111 + 0.001111 = 0.101101|_2 = 0.703125|_{10}$$

natomiast

$$\mathrm{fl}\left((0)1111\ (1)01\right) + \mathrm{fl}\left((0)1111\ (1)10\right) = 0,1010\ |_2 = 0,625\ |_{10}$$

Przykład zastosowania lematu Wilkinsona

Skorzystajmy teraz z lematu Wilkinsona i połóżmy $\varepsilon_1 = \varepsilon_2 = -\frac{1}{9}$. Jako wynik otrzymujemy

$$x_1' + x_2' = 0,1010\ |_2 = 0,625\ |_{10}$$

czyli wynik dokładnie taki sam jak z naszej maszyny wirtualnej. Tutaj mieliśmy

$$|\varepsilon_1| = |\varepsilon_2| < \varepsilon = 2^{-4+1} = \frac{1}{8}$$

Na ogół wybór parametrów ε_1 i ε_2 nie jest jednoznaczny.

Lemat Wilkinsona

Lemat Wilkinsona

Przy odejmowaniu pojawia się czasami efekt z pozoru zaprzeczający lematowi Wilkinsona. Przy odejmowaniu liczb mało różniących się od siebie, różnica ich jest też niewielka, a to może spowodować pojawienie się dużego błędu względnego. Informacja ta jest istotna, np. przy obliczaniu ilorazu różnicowego, w trakcie liczenia pochodnych funkcji, itp. Najprostsze rozwiązanie to zwiększenie dokładności obliczeń maszynowych poprzez wydłużenie słowa, np. liczenie w podwójnej precyzji (DB).

Sumowanie liczb

Aby policzyć sumę n liczb piszemy prosty program komputerowy

```
.................
s=0.0;
for i=1 to n do
s=s+x(i);
.................
```

Analiza wniosków wynikających z lematu Wilkinsona wskazuje, że wyznaczona suma jest dokładną sumą zaburzonych składników, przy czym wielkość zaburzeń zależy od kolejności wykonywania sumowania. Na tej postawie ustanowiono regułę, by *sumować liczby w kolejności wzrastających wartości bezwzględnych*. W pewnych szczególnych przypadkach reguła ta zawodzi, można otrzymać lepszy (dokładniejszy) wynik łamiąc tę regułę, ale wtedy mamy gorsze oszacowanie.

Obliczanie wartości wielomianu

Schemat Hornera

Do obliczania wielomianu

$$w(x) = x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$$

stosujemy tzw. *schemat Hornera*:

$$w_1 = x + a_1\,, \quad w_2 = x w_1 + a_2\,, \quad \ldots \quad w_n = x w_{n-1} + a_n\,,$$

czyli liczymy wyrażenie $w(x) = w_n$

$$x\{x[x\ldots(x + a_1) + a_2] + \ldots + a_{n-1}\} + a_n$$

Obliczanie wartości wielomianu

Schemat Hornera

Korzystając ze schematu Hornera wykonujemy $M = n - 1$ mnożeń i $D = n$ dodawań. Natomiast gdybyśmy wykonali te same obliczenia w sposób naturalny obliczają wielomian zgodnie z jego definicją

$$\underbrace{xx\ldots x}_{n} + a_1 \underbrace{x\ldots x}_{n-1} + \ldots + a_{n-1}x + a_n$$

to trzeba by wykonać $M = \frac{(n-1)(n+2)}{2}$ mnożeń i $D = n$ dodawań. Schemat Hornera jest więc dużo bardziej wydajny. Jest to dosyć istotna informacja, gdyż okazuje się, że oszacowanie błędów zaokrągleń jest identyczne dla obu metod, chociaż wyniki mogą się różnić od siebie.

Spis treści

Obliczenia iteracyjne

Wiele istotnych obliczeń wykorzystuje techniki iteracyjne, polegające na znalezieniu odpowiedniego i odpowiednio zbieżnego ciągu liczbowego przybliżającego poszukiwaną wartość.

Przykład

$$y = \sqrt{x}$$

Budujemy następujący ciąg:

$$y_i = \frac{1}{2}\left(y_{i-1} + \frac{x}{y_{i-1}}\right), \quad i = 1, 2, 3, \ldots$$

gdzie y_0 – dowolnie wybrana liczba dodatnia.

Obliczenia iteracyjne

Jeśli założyć, że obliczenia są wykonywane dokładnie, to ciąg ten jest zbieżny do liczby $\sqrt{x}$. Rzeczywiście, połóżmy $y_i = p_i\sqrt{x}$. Wtedy

$$y_{i+1} = \frac{1}{2}\left(p_i + \frac{1}{p_i}\right)\sqrt{x}$$

a ciąg $p_0 > 0, p_{i+1} = \frac{1}{2}\left(p_i + \frac{1}{p_i}\right)$, $i = 0, 1, 2, \ldots$ jest zbieżny do 1. O tym, czy w kolejnej iteracji nastąpi przybliżenie do rozwiązania decyduje wartość ułamka

$$q_i = \frac{y_{i+1} - \sqrt{x}}{y_i - \sqrt{x}}$$

Obliczenia iteracyjne

Gdy $y_i = p_i\sqrt{x}$, czyli gdy liczymy dokładnie, to

$$q_i = \frac{p_i - 1}{2p_i}$$

a więc $|q_i| < 1$ dla $i = 1, 2, 3, \ldots$, jeśli tylko $p_1 > \frac{1}{3}$.
Gdy wykonujemy te obliczenia na maszynie cyfrowej, to nasz wynik wynosi

$$\text{fl}\,(y_{i+1}) = \frac{1}{2}\left[y_i(1+\varepsilon_1^i) + \frac{x(1+\varepsilon_2^i)(1+\varepsilon_3^i)}{y_i}\right], \quad i = 1, 2, 3, \ldots$$

gdzie $|\varepsilon_k^i| \leq \varepsilon, \quad k = 1, 2, 3, \ldots$.

Obliczenia iteracyjne

Wskutek błędów zaokrągleń okazuje się, że od pewnego momentu otrzymywane rozwiązanie zaczyna oscylować wokół pewnej wartości. Pojawiło się więc u nas nowe pojęcie:

maksymalna graniczna dokładność obliczeń

W tym przypadku oznacza to, że nie możemy zagwarantować lepszego przybliżenia liczby $\sqrt{x}$ niż z błędem

$$\frac{3}{2}\sqrt{x}\,\varepsilon$$

Jest to maksymalna graniczna dokładność wyznaczania liczby $\sqrt{x}$ rozważaną metodą iteracyjną.

Spis treści

Uwarunkowanie zadania i stabilność algorytmów

Zadanie numeryczne jest problemem polegającym na wyznaczeniu *wektora wyników* $\mathbf{w}$ na podstawie *wektora danych* $\mathbf{a}$. Mówimy, że zadanie jest *dobrze postawione*, jeżeli wektor $\mathbf{w}$ jest jednoznacznie określony dla przyjętego wektora danych $\mathbf{a}$. Niech $\mathbf{D}$ oznacza zbiór danych $\mathbf{a}$, dla których zadanie jest dobrze postawione. Na zbiorze $\mathbf{D}$ istnieje zatem odwzorowanie W takie, że

$$\mathbf{w} = W(\mathbf{a})\,. \tag{7}$$

Uwarunkowanie zadania i stabilność algorytmów

Niech $\bar{\mathbf{w}}$ oznacza obliczony numerycznie wektor wyników. W celu obliczenia wektora $\bar{\mathbf{w}}$, należy:

1. sformułować *algorytm obliczeniowy* polegający na określeniu ciągu działań, jakie trzeba wykonać nad wektorem danych $\mathbf{a}$ i wynikami poprzednich działań, (algorytm ten będzie poprawnie sformułowany wtedy, gdy liczba niezbędnych działań będzie skończona (choć może zależeć od wektora danych $\mathbf{a}$)),
2. zrealizować algorytm numerycznie.

Uwarunkowanie zadania i stabilność algorytmów

Algorytm określa odwzorowanie WN takie, że

$$\mathbf{w} = WN(\mathbf{a}, \varepsilon) \tag{8}$$

Parametr ε symbolizuje zależność wektora wyniku $\bar{\mathbf{w}}$ od rodzaju arytmetyki maszyny cyfrowej. Należy podkreślić, że zbiór danych $\mathbf{a}$, na którym odwzorowanie $WN(\mathbf{a}, \varepsilon)$ jest określone, niekoniecznie jest równy zbiorowi $\mathbf{D}$. Oznaczmy ten zbiór przez $\mathbf{DN}$ i załóżmy, że

$$DN \cap D \neq \emptyset\,. \tag{9}$$

Warunek ten odpowiada wymaganiu, by istniały dane $\mathbf{a}$, dla których istnieje rozwiązanie zadania ($\mathbf{a} \in \mathbf{D}$) i można wyznaczyć numeryczne rozwiązanie danym algorytmem ($\mathbf{a} \in \mathbf{DN}$). Rozpatrzmy to na przykładzie.

Uwarunkowanie zadania i stabilność algorytmów

Przykład

Dana jest liczba zespolona $a = x + iy$. Należy obliczyć $\frac{1}{a^2}$.

Algorytm:

- oblicz $t = \frac{y}{x}$ (tangens fazy liczby a)
- oblicz $|a|^2 = x^2 + y^2$ (kwadrat modułu liczby a)
- oblicz $\Re\left(\frac{1}{a^2}\right) = \frac{1}{a^2}\frac{1-t^2}{1+t^2}$, $\Im\left(\frac{1}{a^2}\right) = \frac{1}{a^2}\frac{-2t}{1+t^2}$

Uwarunkowanie zadania i stabilność algorytmów

Dyskusja: Zadanie powyższe jest dobrze postawione jeśli tylko $x^2 + y^2 \neq 0$. Zatem $\mathbf{D} = \mathbb{R}^2 - \{(0,0)\}$. Algorytm jest poprawnie sformułowany (11 niezbędnych działań), lecz nie dla każdej pary $(x, y) \neq 0$ można tym algorytmem znaleźć rozwiązanie. Podczas wykonywania działań mogą bowiem wystąpić zatrzymania pracy maszyny z powodu nadmiaru liczb zmiennopozycyjnych. Będzie tak oczywiście dla wartości $x = 0$, ale nie tylko, bo także dla wartości x o tak małym module, że są zaokrąglane w maszynie do zera. Jeśli ponadto największa liczba zmiennopozycyjna w maszynie ma wartość rzędu 10^{80}, a przyjmiemy $y = 10^{50}$ i $x = 10^{-50}$, to algorytm zatrzyma się z powodu nadmiaru już przy pierwszym działaniu. Widzimy zatem, że $\mathbf{DN} \subset \mathbf{D}$, lecz $\mathbf{DN} \neq \mathbf{D}$.

Uwarunkowanie zadania i stabilność algorytmów

Zwróćmy uwagę, że zbiór **D** nie zależy ani od dokładności działań wykonywanych przez maszynę cyfrową, ani od zakresu liczb, jakie można wprowadzić do rejestru maszyny. Zbiór ten charakteryzuje bowiem jedynie dane zadanie. Inaczej jest w przypadku zbioru **DN**: jest on zależny od powyższych własności maszyny cyfrowej. Dla podkreślenia tego, będziemy oznaczać ten zbiór symbolem $\mathbf{DN}(\varepsilon)$. Przyjmiemy poza tym umowę, że jeśli zmieniamy własności arytmetyki maszyny tak, że ε maleje, to równocześnie wzrasta zakres liczb, które można wprowadzać do rejestrów maszyny bez sygnalizowania nadmiaru. Założenie to jest zgodne z najczęściej stosowaną metodą zmniejszania ε (zwiększania dokładności), polegającą na wydłużaniu słów maszynowych.

Koniec? :-(

Koniec wykładu 1