Zapis zmiennopozycyjny, arytmetyka, błędy numeryczne

Plan wykładu:

1. zapis zmiennopozycyjny
2. arytmetyka zmiennopozycyjna
3. reprezentacja liczb w standardzie IEEE754
4. błędy w obliczeniach numerycznych
5. definicje (zadanie numeryczne, algorytm numeryczny,...)

Własności zapisu zmiennopozycyjnego

Liczbę rzeczywistą w komputerze reprezentuje liczba zmiennoprzecinkowa:

$$F = M\beta^E$$

M – znacznik („mantysa”) jest liczbą ułamkową ze znakiem
β – stanowi bazę reprezentacji i jest liczbą całkowitą (np.: 2, 10, 16)
E – wykładnik („cecha”) jest znakowaną liczbą całkowitą

Powyższy zapis jest niejednoznaczny:

$$F = M\beta^E = M_i\beta^{E+i} = (M\beta^{-i})\beta^{E+i}$$
$$i = -m, \ldots, -1, 0, 1, \ldots, m$$

m jest liczbą pozycji znacznika w bazie.
Można odróżnić

$$k = m \log_\beta 2$$

różnych reprezentacji tej samej liczby.

Ograniczeniem znacznika jest

$$|M| < \beta$$

ale problem nadal pozostaje.

Jednoznaczność osiągamy dla warunku

$$\beta^{p-1} \leq |M| < \beta^p$$

wtedy dla dowolnego i≠ 0 mamy

$$\beta^{p-i-1} \leq |M|\beta^{-i} < \beta^p \qquad\qquad |M|\beta^{-i} \notin [\beta^{p-1}, \beta^p)$$

w praktyce stosuje się

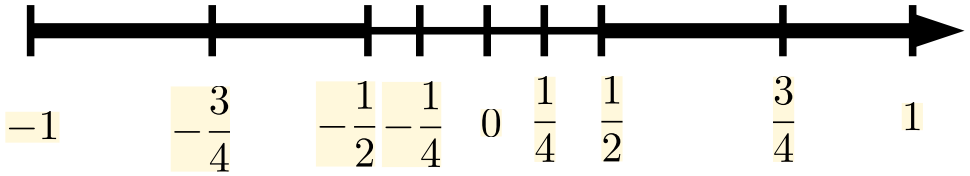
$$p = 0, 1$$

Znaczniki spełniające powyższy warunek nazywamy **znormalizowanymi (liczby znormalizowane)**.

Nie ma liczb znormalizowanych

$$|F| < \beta^{p-1-E_{min}}$$

w tym 0. Tworzą one osobną grupę zwana **liczbami zdenormalizowanymi**.



Znormalizowane wartości dla $\beta = 2, \quad p = 0$

Podstawowe operacje arytmetyczne.

Wykonujemy operacje na dwóch znormalizowanych liczbach

$$\mathrm{F}_1 = M_1\beta^{E_1}, \quad F_2 = M_2\beta^{E_2}$$

Czy przeprowadzenie 4 podstawowych operacji da znormalizowany wynik?

Mnożenie

$$\begin{aligned} F_1F_2 &= M_1\beta^{E_1} \cdot M_2\beta^{E_2} \\ &= (M_1M_2)\beta^{E_1+E_2} = M_W\beta^W \end{aligned}$$

sprawdzamy czy nie wystąpił **nadmiar** ($E_W>E_{max}$) lub **niedomiar** ($E_W<E_{min}$) zmiennopozycyjny.

Jeśli wartość znacznika wychodzi poza dozwolony zakres tj.:

$$\beta^{2p-2} \leq |M_W| = |M_1M_2| < \beta^{p-1}$$

to wykonujemy **postnormalizację**

$$\begin{aligned} F_1F_2 &= (M_1M_2\beta^{-p+\varepsilon})\beta^{E_1+E_2+p-\varepsilon} \\ \varepsilon &= 0,1 \end{aligned}$$

Dzielenie

$$\frac{F_1}{F_2} = \frac{M_1 \beta^{E_1}}{M_2 \beta^{E_2}} = \frac{M_1}{M_2} \beta^{E_1 - E_2}$$

ponieważ

$$|M_W| = \frac{M_1}{M_2} \in (\beta^{-1}, \beta)$$

konieczna jest postnormalizacja ilorazu do postaci

$$F_1/F_2 = (\beta^{p-\varepsilon} M_1/M_2)\beta^{E_1 - E_2 - p + \varepsilon}$$
$$\beta^{-1} \leq |M_W| < 1 \Longrightarrow \varepsilon = 0$$
$$1 \leq |M_W| < \beta \Longrightarrow \varepsilon = 1$$

Dodatkowo należy zapewnić obsługę:
1) liczb zdenormalizowanych,
2) dzielenia przez 0

Dodawanie i odejmowanie

Wymagane jest wstępne wyrównanie wykładników.
Powoduje to denormalizację operandu z mniejszym wykładnikiem
i utratę dokładności (na najmniej znaczących pozycjach znacznika).

Załóżmy

$$\mathrm{E}_1 \geq E_2$$

$$\begin{aligned} F_1 \pm F_2 &= \left(M_1 \beta^{E_1}\right) \pm \left(M_2 \beta^{E_2}\right) \\ &= \left(M_1 \pm M_2 \beta^{-(E_1 - E_2)}\right) \beta^{E_1} \end{aligned}$$

a) jeśli

$$\mathrm{E}_1 = E_{max}$$

oraz

$$\beta^p \leq |M_W| < 2\beta^p$$

to wówczas normalizacja doprowadzi do **nadmiaru**.

b) jeśli

$$|M_W| < \beta^i \leq \beta^{p-1}$$

oraz

$$\mathrm{E}_1 - (p - i) \leq E_{min}$$

to wystąpi **niedomiar**.

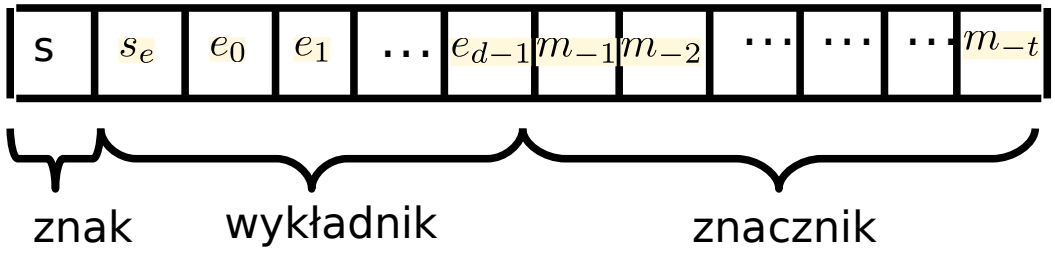
Wybór reprezentacji

Liczbę zmiennopozycyjną zapisujemy na **n** pozycjach, z czego:

- **1** pozycję przeznaczamy na znak liczby
- **t** pozycji na zapis znacznika
- **d** pozycji na zapis wykładnika

$$n = 1 + d + t$$

$$x = s\left(\sum_{i=1}^{t} m_{-i}\beta^{-i}\right)\beta^{E}$$



Zaokrąglanie do najbliższej wartości

- reguły zaokrąglania

$$R(x) = \begin{cases} M, & x - M < \frac{ulp}{2} \\ M + ulp, & x - M \geq \frac{ulp}{2} \end{cases}$$

- ten typ zaokrąglania powoduje, że średnia wartość błędu standaryzowanego jest bliska 0, **ale estymator R(x) jest obciążony dodatnio**

Zaokrąglanie symetryczne

- reguły zaokrąglania

$$S(x) = \begin{cases} M - ulp, & -ulp \leq x - M < -\frac{1}{2}ulp \\ M, & -\frac{1}{2}ulp \leq x - M \leq +\frac{1}{2}ulp \\ M + ulp, & +\frac{1}{2}ulp \leq x - M < +ulp \end{cases}$$

- średnia wartość błędu standaryzowanego wynosi 0, a estymator S(x) jest niebciążony
- wadą zwykłego zaokrąglania oraz zaokrąglania symetrycznego jest konieczność wykonania 2m-pozycyjnego dodawania

Dokładność reprezentacji

- jeśli liczba zmiennopozycyjna jest reprezentowana przez skończoną liczbę bitów
  to dokładność reprezentacji określa liczba bitów znacznika
  a zakres reprezentowanych liczb zależy od liczby bitów wykładnika

Jeśli zachodzi warunek

$$M\beta^E \leq x \leq (M + ulp)\beta^E, \quad x \in R$$

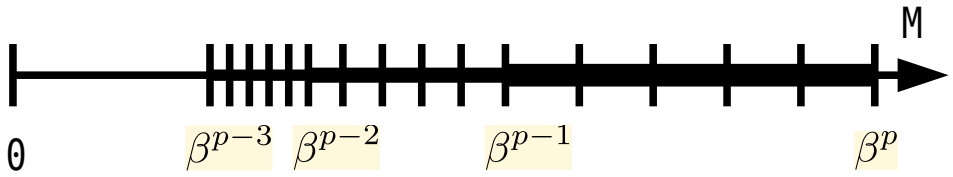
(**ulp** – najmniej znacząca pozycja znacznika)

wówczas zaokrąglając liczbę x do bliższej reprezentacji dostajemy **błąd bezwzględny**

$$|fl(x) - x| \leq \frac{1}{2} ulp \beta^E$$

gdzie: **fl(x)** oznacza reprezentację liczby x w zapisie zmiennopozycyjnym.

- wartość błędu bezwzględnego wynika z nierównomiernego rozłożenia liczb w reprezentacji zmiennopozycyjnej



Błąd względny

$$|\varepsilon(x)| = \frac{|fl(x) - x|}{x} \leq \frac{\frac{1}{2}ulp\beta^E}{M\beta^E} = \frac{ulp}{2M}$$

rośnie ze zmniejszaniem znacznika

Dobór zakresu wykładnika

- jednym z wymagań jest aby dla każdej znormalizowanej liczby x możliwe było obliczenie jej odwrotności
- wykładnik kodowany jest na d pozycjach, z czego λ pozycji musimy zarezerwować na **0** oraz **wielkości nieznormalizowane** (nieskończoności itp.).
- rozpiętość wykładnika wynosi

$$s = E_{max} - E_{min} = 2^d - \lambda$$

tak aby istniały znormalizowane reprezentacje odwrotności małych liczb.

- przykład: β=2, β-1=1 i p=0,

$0,1000000$ - najmniejszy znacznik

$0,1111111$ - najwiekszy znacznik

- najbardziej znaczący bit nie musi być kodowany – tzw. **bit ukryty**
- pominięcie go prowadzi do pełnego wykorzystania przestrzeni kodowej znacznika (100%)
- w innych przypadkach liczby znormalizowane zajmują jedynie (1-1/β)100% przestrzeni
- bit ukryty jest odtwarzany w trakcie operacji arytmetycznych.

Schematy zaokrąglania liczb

- podczas wykonywania operacji arytmetycznych może dojść do zwiększenia się liczby bitów wynikowych, np. przy mnożeniu dwóch znaczników

$$m_w = 2m$$

- aby zapisać wynik należy uciąć ostanie m bitów
- eliminacja nadmiarowych bitów nazywana jest **zaokrąglaniem**
- reguły zaokrąglania

$$x \leq y \Rightarrow fl(x) \leq fl(y)$$
$$x \in fl \Rightarrow fl(x) = x$$
$$\underbrace{F_1 = M\beta^E \leq x \leq (M + ulp)\beta^E = F_2}_{x=F_1 \vee x=F_2}$$

Odcięcie (najprostsze)

$$M \leq x < M + ulp \Leftrightarrow T(x) = M$$

- jeśli: m - liczba pozycji znacznika, d - liczba uciętych bitów

$$x = M + i2^{-d}ulp, \quad 0 \leq i \leq 2^d - 1$$

wówczas standaryzowanym błędem losowym obcinania znacznika jest

$$\delta_T = \frac{T(x) - x}{ulp} = -i2^{-d}$$

- dla równomiernego rozkładu wartości x w przedziale [M, M+ulp] miarą średniego standaryzowanego błędu obcinania jest

$$\langle \delta_T \rangle = \frac{1}{2^d} \sum_{i=0}^{2^d-1} (-i)2^{-d} < 0$$

błąd ten jest zawsze ujemny – estymator T(x) jest ujemnie obciążony

- efekt obcinania: **niedoszacowanie**.

Reprezentacja liczb w standardzie IEEE754

- formaty liczb zmiennopozyjcyjnych:
  - zwykły pojedynczej precyzji - **single (real)**
  - rozszerzony pojedynczej precyzji - **single extended**
  - zwykły podwójnej precyzji – **double**
  - rozszerzony podwójnej precyzji - **double extended**
- wykładnik jest reprezentowany w kodzie z obciążeniem, a znacznik w kodzie znak-moduł
- wartość liczby w IEEE754

$$F = (-1)^s 2^{E_B - B}(1, f)$$

E_B - wykładnik,
B – przesunięcie (dzięki niemu nie musimy pamietać znaku),
($E=E_B$-B -"prawdziwy" wykładnik)
(1,f) – wartość modułu znacznika

jeśli d oznacza liczbę bitów wykładnika to wielkością przemieszczenia jest

$$B = 2^d - 1$$

- bez przesunięcia, najmniejszą wartością wykładnika jest

$$E_B = E_{min} + B = 00\ldots01_2$$

a największą

$$E_B = E_{max} + B = 11\ldots11_2$$

obie liczby dodatnie

- zakres wykładnika jest ograniczony z obawy przed uzyskaniem nadmiaru podczas obliczania odwrotności liczb

Format zapisu liczb zmiennopozycyjnych w IEEE 754

	symbol	single	double
Rozmiar formatu	n	32	64
Rozmiar znacznika	m	23(+1)	52(+1)
Rozmiar wykładnika	d	8	11
obciążenie	B	127	1023
Zakres wykładnika	E	[-126,127]	[-1022,1023]
dokładność	ulp	$2^{-23} \approx 10^{-7}$	$2^{-52} \approx 10^{-15}$
Zakres formatu	RNG	$\approx 2^{128}$ $\approx 3,8 \cdot 10^{38}$	$\approx 2^{1024}$ $\approx 9 \cdot 10^{307}$

Nie-liczby

- pojawiają się podczas wykonywania operacji, np.:

$$0/0, \quad \sqrt{-|x|}$$

bez ich obsługi program przerywałby działanie, a to jest niepożądane

- obsługa nie-liczb pozwala je wykryć i np. zrestartować schemat iteracyjny z innym parametrem

Znakowane zero

- po co? dla liczb rzeczywistych mamy

$$1/-\infty = 0, \qquad 1/(1/-\infty) = +\infty$$

zatem relacja

$$1/(1/x) = x$$

nie jest spełniona

- dla znakowanego 0 mamy

$$x = -\infty, \; 1/x = -0$$

$$1/(1/x) = 1/-0 = -\infty = x$$

Wyjątki w IEEE754

standard zapewnia obsługę specyficznych wyników operacji:

- nadmiar (F_{max}, nieskończoność)
- niedomiar (F_{min}, l. denormalizowane)
- dzielenie przez 0
- niepoprawna operacja (NAN – *not a number*)
- niedokładność (zaokrąglenie wyniku)

Błędy numeryczne

- najprostszy podział:

  1. błędy wejściowe
  2. błędy zaokrągleń
  3. błędy obcięcia

Błędy wejściowe

- występują, gdy dane liczbowe wprowadzane do pamięci komputera odbiegają od wartości dokładnych
- w szczególności:
  - gdy wprowadzane dane pomiarowe są obarczone błędami pomiarowymi (np. pomiar wielkości fizycznych takich jak oporu czy napięcia)
  - gdy ze względu na skończoną długość słowa binarnego dochodzi do wstępnego zaokrąglenia liczb (ułamki dziesiętne lub zaokrąglanie liczb niewymiernych jak np.: e, π)

Przykład – zapis 8 bitowy

Liczba

$$x_{(10)} = 3.25$$

ma reprezentację

$$x_{(z2)} = \underbrace{(0)1101}_{M}\underbrace{(0)10}_{W}$$

Ale dla liczby **x=0.2** pojawia się problem

$$x_{(2)} = 0.0011(0011)\ldots$$

po zaokrągleniu wyniku do najbliższej liczby

$$x^{'}_{(z2)} = (0)1100(1)10$$
$$x^{'}_{(2)} = 0.001100$$
$$x^{'}_{(10)} = 0.1875$$

co daje błąd bezwzględny równy 0.0125 i błąd względny na poziomie 6.25%.

Błędy obcięcia

- powstają podczas zmniejszania liczby działań np.:

  a) przy obliczaniu wartości szeregów (ucięcie szeregu)

  b) wyznaczaniu granic (obliczanie wartości całki)

  c) zastępowaniu pochodnej funkcji ilorazem różnicowym

Przykład - należy wyznaczyć wartość e^x

korzystamy z rozwinięcia

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \quad -\infty < x < +\infty$$

ale numerycznie lepiej zrobić to „nieco inaczej”

$$e^x = e^{E(x)} e^q, \qquad 0 \leq q < 1$$

gdzie: E(x) jest częścią całkowitą liczby x, q jest częścią ułamkową

- pierwszy wyraz jest potęgą, a drugi liczymy wg rozwinięcia
- do wyznaczenia pozostaje tylko błąd obcięcia – bo szereg musimy „gdzieś” uciąć
- szereg ucinamy na n-tym wyrazie (jakie przyjąć n?).

Reszta szeregu (uwzględniamy n wyrazów):

$$R_n(x) = \frac{e^{\theta q}}{(n+1)!} q^{n+1}, \quad 0 < \theta < 1$$

wnioski:

- licznik jest ograniczony < e~2.73
- mianownik rośnie ze względu na silnię
- szereg jest szybkozbieżny

$$\begin{aligned}
R_n(x) = R_n(q) &= \frac{q^{n+1}}{(n+1)!} + \frac{q^{n+2}}{(n+2)!} + \frac{q^{n+3}}{(n+3)!} + \dots \\
&= \frac{q^{n+1}}{(n+1)!}\left(1 + \frac{q}{n+2} + \frac{q^2}{(n+2)(n+3)} + \dots\right) \\
&< \frac{q^{n+1}}{(n+1)!}\left(1 + \frac{q}{n+2} + \left(\frac{q}{n+2}\right)^2 + \dots\right) \quad \text{- szereg geometryczny} \\
&< \frac{q^{n+1}}{(n+1)!}\frac{1}{1-\frac{q}{n+2}} \approx \frac{q^{n+1}}{(n+1)!}\frac{n+2}{n+1} \\
&< \frac{q^{n+1}}{(n+1)!}\frac{n+1}{n}
\end{aligned}$$

$$\frac{n+2}{n+1} < \frac{n+1}{n}$$

$$0 < R_n(q) < \frac{q^n}{n!}\frac{q}{n} = u_n \frac{q}{n}, \qquad 0 < q < 1$$

Jak liczymy wartość szeregu?

- sumując kolejne wyrazy, aż R_n stanie się akceptowalnie małe

$$e^x = u_0 + u_1 + u_2 + \ldots + R_n(x)$$

- stosujemy schemat rekurencyjny

$$u_k = \frac{x}{k} u_{k-1}$$
$$s_k = s_{k-1} + u_k$$
$$k = 0, 1, 2, \ldots, n$$

$$R_k = u_k \frac{x}{k} < \varepsilon ?$$

← do momentu aż ten warunek zostanie spełniony

parametry startowe: $u_0 = 1, \quad s_{-1} = 0$

Przykład. Obliczmy wartość $\sqrt{e}$ z dokładnością 2.5x10^{-6}

$$
\begin{aligned}
u_0 &= 1 \\
u_1 &= 0.5000000 \\
u_2 &= 0.1250000 \\
u_3 &= 0.0208333 \\
u_4 &= 0.0026042 \\
u_5 &= 0.0002604 \\
u_6 &= 0.0000217 \\
u_7 &= 0.0000016 < \varepsilon
\end{aligned}
$$

$$e^{\frac{1}{2}} \approx u_0 + u_1 + u_2 + \ldots + u_7 = 1.648721$$

Błędy zaokrągleń

- pojawiają się podczas wykonywania operacji arytmetycznych
- wynikają z ograniczonej reprezentacji liczb zmiennopozycyjnych
- wielkość błędów zależy od:

  a) dokładności reprezentacji

  b) sposobu zaokrąglania wyniku

  c) rodzaju przeprowadzanej operacji

Lemat Wilkinsona – błędy zaokrągleń powstające podczas wykonywania działań zmiennopozycyjnych są równoważne zastępczemu zaburzeniu liczb, na których wykonujemy działania.

- zaburzone liczby możemy zapisać w użytecznej postaci

$$fl(x) = x(1+\varepsilon_x), \quad fl(y) = y(1+\varepsilon_y), \qquad \varepsilon_x, \varepsilon_y << 1$$

Błędy względne zaokrągleń

- mnożenia

$$\frac{fl(x)fl(y) - xy}{xy} = (1+\varepsilon_x)(1+\varepsilon_y) - 1 = \varepsilon_x + \varepsilon_y + \varepsilon_x\varepsilon_y \approx \varepsilon_x + \varepsilon_y$$

- dzielenia

$$\frac{fl(x)/fl(y) - x/y}{x/y} = \frac{1+\varepsilon_x}{1+\varepsilon_y} - 1 = \frac{\varepsilon_x - \varepsilon_y}{1-\varepsilon_y} \approx \varepsilon_x - \varepsilon_y$$

- dodawania i odejmowania

$$\frac{fl(x) \pm fl(y) - (x \pm y)}{x \pm y} = \frac{x\varepsilon_x \pm y\varepsilon_y}{x \pm y} = \frac{x}{x \pm y}\varepsilon_x - \frac{y}{x \pm y}\varepsilon_y$$

- zwłaszcza przy odejmowaniu możemy dostać duży błąd, gdy

$$x \approx y, \quad \varepsilon_x \approx -\varepsilon_y$$

ze względu na kasowanie mianownika

Wykonywanie kolejnych operacji na wynikach poprzednich operacji prowadzi do kumulacji błędów zaokrągleń

- to pesymistyczny scenariusz, ale tego należy oczekiwać podczas obliczeń numerycznych

Błędy można zmniejszyć:

1) ustalając odpowiednio sposób i kolejność wykonywanych działań (np. algorytm Kahana dla iloczynu skalarnego),

2) zwiększając precyzję obliczeń (nie zawsze można – naukowe i inżynierskie w zasadzie zawsze wykonujemy w podwójnej precyzji),

3) stosując inny algorytm implementujący daną metodę

(1) w zasadzie nie stosuje się – sortowanie jest czasochłonne
(2) używamy już silnej arytmetyki
(3) najbardziej użyteczny sposób – zmiana algorytmu

Przykłady szacowania błędów zaokrągleń

a) Sumowanie liczb (jedna z częściej wykonywanych operacji)

$$s = \sum_{i=1}^{n} x_i$$

oznaczenie $\quad s_k = fl\left(s_{k-1} + x_k\right) = s_{k-1}^{'} + x_k^{'}$

zgodnie z lematem Wilkinsona:

$$s_{k-1}^{'} = s_{k-1}(1+\varepsilon_{k-1}^{s}) \qquad |\varepsilon_{k-1}^{s}| \leq \varepsilon$$

$$x_k^{'} = x_k(1+\varepsilon_k^{x}) \qquad |\varepsilon_k^{x}| \leq \varepsilon$$

Indeks s - suma
indeks x - wartość

liczymy wartość sumy sumując kolejne wkłady:

$$\begin{aligned} s = {} & x_1(1+\varepsilon_1^x)(1+\varepsilon_2^s)\cdots(1+\varepsilon_{n-1}^s) \\ & + x_2(1+\varepsilon_2^x)(1+\varepsilon_2^s)\cdots(1+\varepsilon_{n-1}^s) \\ & + x_3(1+\varepsilon_3^x)(1+\varepsilon_3^s)\cdots(1+\varepsilon_{n-1}^s) + \cdots \\ & + x_{n-1}(1+\varepsilon_{n-1}^x)(1+\varepsilon_{n-1}^s) + x_n(1+\varepsilon_n^x) \end{aligned}$$

najbardziej zaburzony jest pierwszy wyraz

najmniej zaburzony jest ostatni wyraz

b) Obliczanie wartości wielomianu

$$w(x) = x^n + a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1} x + a_n$$

- w „tradycyjny” ale nieoptymalny sposób:

$$w(x) = \underbrace{x x \cdots x}_{n} + a_1 \underbrace{x \cdots x}_{n-1} + \cdots + a_{n-1} x + a_n$$

M operacji mnożenia,

$$M = \frac{(n-1)(n+2)}{2}$$

D operacji dodawania

$$D = n$$

- optymalny sposób obliczania wartość wielomianu zapewnia **Schemat Hornera**

$$w(x) = x\Big(x\Big(x \cdots \big(x + a_1\big) + a_2\Big) + \cdots + a_{n-1}\Big) + a_n$$

wykonujemy tylko: **M=n-1** mnożeń i **D=n** dodawań

- wyniki uzyskane wg powyższych algorytmów mogą się różnić, natomiast oszacowana największa możliwa wartość błędu w obu przypadkach jest taka sama

zadanie numeryczne

- to jasny i niedwuznaczny opis powiązania funkcjonalnego między danymi wejściowymi i danymi wyjściowymi, dane te składają się ze skończonej liczby wielkości rzeczywistych

algorytm numeryczny

- dla zadania numerycznego to opis poprawnie określonych operacji (arytmetycznych lub logicznych), które należy wykonać aby przekształcić wektor danych wejściowych w wektor danych wyjściowych

Przykład. Określić największy pierwiastek rzeczywisty równania

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

dla wektora danych wejściowych (a_0, a_1, a_2, a_3)

- jest to zadanie numeryczne, dane wyjściowe to szukany pierwiastek
- algorytm numeryczny dla tego zadania to np.: metoda Newtona

uwarunkowanie zadania

wektor danych wejściowych $\boldsymbol{d} = (d_1, d_2, \ldots, d_n) \in R^n$

przekształcamy w wektor wynikowy $\boldsymbol{w} = (w_1, w_2, \ldots, w_m) \in R^m$

stosując określone odwzorowanie $\boldsymbol{w} = \varphi(\boldsymbol{d}), \quad \varphi : R^n \to R^m$

- jeśli niewielkie względne zmiany danych zadania powodują duże względne zmiany rozwiązania, to zadanie takie jest źle uwarunkowane

wskaźnik uwarunkowania - miara uwarunkowania zadania

- nazywamy tak wielkość charakteryzującą wpływ zaburzeń danych wejściowych na wynik

$$\|\varphi(\boldsymbol{d}) - \varphi(\tilde{\boldsymbol{d}})\| = cond(\varphi, \boldsymbol{d})\|\boldsymbol{d} - \tilde{\boldsymbol{d}}\| \qquad \tilde{\boldsymbol{d}} \text{ - dane zaburzone}$$

$cond(\varphi, \boldsymbol{d}) \approx 1$ - zadanie dobrze uwarunkowane (mały wpływ zaburzeń)

$cond(\varphi, \boldsymbol{d}) \gg 1$ - zadanie źle uwarunkowane (duży wpływ zaburzeń)

$cond(\varphi, \boldsymbol{d}) \to +\infty$ - zadanie źle postawione (problemu nie rozwiążemy)

Przykład Jakie jest uwarunkowanie obliczania iloczynu skalarnego?

$$S = \vec{a} \cdot \vec{b} = \boldsymbol{a}^T \boldsymbol{b} = (\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} a_i b_i \neq 0$$

zaburzamy dane wejściowe

$$a_i(\alpha) = a_i(1+\alpha_i), \qquad b_i(\beta) = b_i(1+\beta_i), \qquad \alpha_i \beta_i \approx 0$$

i liczmy względną zmianę wyniku

$$\left| \frac{\sum_{i=1}^{n} a_i(1+\alpha_i) b_i(1+\beta_i) - \sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a_i b_i} \right| \approx \left| \frac{\sum_{i=1}^{n} a_i b_i (\alpha_i + \beta_i)}{\sum_{i=1}^{n} a_i b_i} \right| \leq \max_i |\alpha_i + \beta_i| \left(\frac{\sum_{i=1}^{n} |a_i b_i|}{\left| \sum_{i=1}^{n} a_i b_i \right|} \right)$$

- wskaźnik uwarunkowania przyjmujemy w postaci $cond(\boldsymbol{a}, \boldsymbol{b}) = \frac{\sum_{i=1}^{n} |a_i b_i|}{|\sum_{i=1}^{n} a_i b_i|}$
- zadanie będzie dobrze uwarunkowane jeśli $|a_i b_i| = a_i b_i \rightarrow cond(\boldsymbol{a}, \boldsymbol{b}) = 1$
- zadanie może być źle uwarunkowane, gdy wektory będą „prawie” ortogonalne

stabilność algorytmu numerycznego

potoczna definicja stabilności:
jeśli różnica między rozwiązaniem dokładnym a numerycznym pozostaje skończona

$$\|\varphi(\boldsymbol{d}) - fl(A(\boldsymbol{d}))\| \leq K = const$$

$fl(A(\boldsymbol{d}))$ - realizacja odwzorowania przez algorytm A
w arytmetyce zmiennopozycyjnej

- stabilność jest własnością jakiej wymagamy od algorytmu numerycznego
- jeśli wiemy, że dla danej klasy zadań i/lub danych wejściowych algorytm jest niestabilny, to zmieniamy dane wejściowe (jeśli to pomoże) lub zmieniamy algorytm na inny (stabilny)

złożoność obliczeniowa algorytmu numerycznego

- wykonując obliczenia numeryczne na komputerze przeprowadzamy ciąg kolejnych operacji arytmetycznych, a to przekłada się na czas wykonania algorytmu

- złożoność obliczeniowa algorytmu określa liczbę operacji jaką należy wykonać w celu uzyskania wyniku

- w praktycznych zastosowaniach czas wykonywanych obliczeń jest kluczowy, mając dwa algorytmy realizujące to samo zadanie, wybierzemy ten który ma mniejszą złożoność obliczeniową

  przykład już mieliśmy – sposób obliczania wartości wielomianu

- złożoność obliczeniowa to nie jedyny czynnik wpływający na czas wykonania obliczeń, często zależy także od sposobu ich wykonania, co zależy od

  1) architektury maszyny (CPU ↔ GPU)

  2) kolejności wykonywanych obliczeń ze względu na szybkość dostępu do danych w pamięci komputera: RAM → CPU → RAM

- optymalizacja kodu (numerycznego) to osobne (szerokie) zagadnienie i nie będziemy się nim zajmować