Metody Obliczeniowe w Nauce i Technice

Wprowadzenie, arytmetyka komputerowa

Marian Bubak, Katarzyna Rycerz

Department of Computer Science
AGH University of Science and Technology
Krakow, Poland
kzajac@agh.edu.pl
dice.cyfronet.pl

Contributors
Maciej Trzebiński
Mikołaj Biel
Rafał Stachura

Outline

Wprowadzenie

- metody numeryczne to sposoby rozwiązywania złożonych problemów matematycznych za pomocą podstawowych operacji arytmetycznych,
- wykorzystywane gdy badany problem:
  - nie ma w ogóle rozwiązania analitycznego (danego wzorami)
  - obliczenie na podstawie wzoru otrzymanego analitycznie ma dużą złożoność
  - obliczenie na podstawie wzoru otrzymanego analitycznie jest źle uwarunkowane numerycznie
- otrzymywane wyniki są na przybliżone,
- dokładność obliczeń może być z góry określona i dobiera się ją zależnie od potrzeb.

Literatura

- teoria: D. Kincaid, W. Cheney, Analiza numeryczna
- praktyka: Piotr Krzyżanowski, Obliczenia inżynierskie i naukowe
- prosty poradnik: W. Kordecki, K. Serwat, Metody numeryczne dla informatyków

Reprezentacja stałopozycyjna (integer)

- np. kod U2: na d+1 bitach reprezentowane dokładnie liczby

$$L \in [-2^d, 2^d - 1]$$

Uzupełnieniem dwójkowym liczby x zapisanej za pomocą n bitów nazywamy liczbę równą $2^n - x_{(U2)}$

- gdy argumenty i wynik reprezentowane stałopozycyjnie, to działania na nich: $+$, $-$, $\cdot$, $/(div)$, $/(mod)$ są wykonywane dokładnie

Zastosowania:

- sprawy walutowe, operacje monetarne
- procesory graficzne np. Sony Nintendo
- rozmiary czcionek w calach np. w TeX
- libfixmath - implementacja biblioteki stałoprzecinkowej w C

Reprezentacja zmiennoprzecinkowa (float)

Należy pamiętać o ułomności reprezentacji zbioru liczb rzeczywistych $\mathbb{R}$ w rzeczywistym świecie skończonych komputerów.

F - zbiór liczb zmiennoprzecinkowych (floating-point):

β - podstawa,
t - dokładność,
L, U - zakres wykładnika

d_i - liczby całkowite,
$0 \leqslant d_i \leqslant \beta - 1, i = 1, \ldots, t$
$L \leqslant e \leqslant U$

$x \in F$ ma wartość:

$$x = \pm \underbrace{\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t} \right)}_{\text{mantysa}} \cdot \beta^{\overbrace{e}^{\text{cecha}}} = \pm \sum_{i=1}^{t} \frac{d_i}{\beta^i} \cdot \beta^e$$

System F jest *unormowany*, gdy $\forall_{x \neq 0}\ d_1 \neq 0$.

Reprezentacja zmiennoprzecinkowa (float)

Ważne

F nie jest kontinuum - więcej - jest skończony o liczbie elementów wyrażonych wzorem:

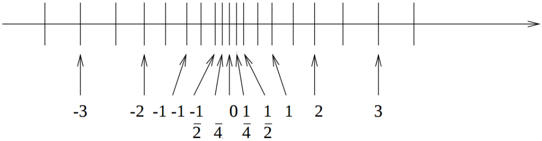
$$2 \cdot (\beta - 1) \cdot \beta^{t-1} \cdot (U - L + 1) + 1$$

Wyjaśnienie:

- $2 \rightarrow$ znak liczby $\pm$
- $(\beta - 1) \rightarrow$ liczba możliwości na pierwszym bicie mantysy (o jeden mniej, bo unormowanie – nie ma zera)
- $\beta^{t-1} \rightarrow$ liczba możliwości na pozostałych t-1 bitach
- $(U - L + 1) \rightarrow$ zakres wykładnika
- $1 \rightarrow$ zero

Reprezentacja zmiennoprzecinkowa (float)

Elementy F nie są równomiernie rozłożone na osi:
$\beta = 2, t = 3, L = -1, U = 2$ (33 elementy):



Każdy element F reprezentuje cały przedział liczb $\mathbb{R}$
x - l. rzeczywista $\in$ zakresu F,
$fl(x)$ - reprezentacja zmiennoprzecinkowa liczby x

$$\left|\frac{fl(x) - x}{x}\right| \leqslant \frac{1}{2}\beta^{1-t}$$

β^{1-t} - oszacowanie względnej dokładności arytmetyki

Zadanie: sprawdzić

Reprezentacja zmiennoprzecinkowa (float)

Uwaga

0.1 - często krok w algorytmach
Czy 10 kroków o długości 0.1 to to samo co 1 krok = 1.0?

W systemie o podstawie $\beta = 2$ - **nie!**

$$0.1_{10} = 0.0(0011)_2 = 0.0(12)_4 = 0.0(6314)_8 = 0.199999\ldots_{16}$$

Reprezentacja 0.1 urywa się po t cyfrach. Dodanie 10 tak uzyskanych liczb nie da dokładnie 1.0.

Porównania w arytmetyce float

Zamiast przyrównywać wartości należy sprawdzać, czy otrzymany błąd pomiędzy wartością obliczoną, a oczekiwaną jest mniejszy od zadanego ϵ.

Błąd reprezentacji zmiennoprzecinkowej

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

s ← sign, c← cecha, m ← mantysa

$$m = \sum_{i=1}^{\infty} e_i \cdot 2^{-i}$$

$$e_i = \begin{cases} 0 \\ 1 \end{cases}$$

Reprezentacja mantysy

$$m_t = \underbrace{\sum_{i=1}^{t} e_i \cdot 2^{-i}}_{\text{t-bitowa mantysa}} + \underbrace{e_{t+1} \cdot 2^{-t}}_{\text{zaokrąglenie dodajemy do ostatniego bitu reprezentacji}}$$

Błąd reprezentacji zmiennoprzecinkowej

a)Jeśli liczba jest bliżej do swojego zaokrąglenia w dół to po prostu pomijamy resztę bitów i reprezentacja jest równa :

$$fl(x)^{-} = \pm \sum_{i=1}^{t} \frac{d_i}{\beta_i} \cdot \beta^e$$

Najdalsza "końcówka" takiej liczby to:

t	t+1	t+2	...	
	0	1	...	1

$$m = \underbrace{\sum_{i=1}^{t} e_i \cdot 2^{-i} + 0 \cdot 2^{-(t+1)}}_{\text{to samo co } m_t = \sum_{i=1}^{t} e_i \cdot 2^{-i} + 0 \cdot 2^{-t}} + \underbrace{\sum_{i=t+2}^{\infty} 1 \cdot 2^{-i}}_{\text{suma szeregu: } \frac{1}{2^{t+1}} = 2^{-(t+1)}}$$

$$m - m_t = 2^{-(t+1)}$$

Błąd reprezentacji zmiennoprzecinkowej

b)Jeśli liczba jest bliżej do swojego zaokrąglenia w górę to dodajemy 1 do ostatniego bitu i reprezentacja jest równa :

$$fl(x)^{+} = \pm(\sum_{i=1}^{t} \frac{d_i}{\beta_i} + \frac{1}{\beta_t}) \cdot \ \beta^{e}$$

Najdalsza "końcówka" takiej liczby to:

t	t+1	t+2	...	
	1	0	...	0

$$m = \sum_{i=1}^{t} e_i \cdot 2^{-i} + 2^{-(t+1)}$$

$$m_t = \sum_{i=1}^{t} e_i \cdot 2^{-i} + 2^{-t}$$

$$m_t - m = 2^{-(t+1)}$$

Błąd reprezentacji zmiennoprzecinkowej

W obydwu przypadkach maksymalny błąd bezwzględny to:

$$m_t - m = 2^{-(t+1)}$$

W przyjętej reprezentacji ($d_1 = 1$, pozostałe bity dowolne) najmniejsze m to $\frac{1}{2}$.
Błąd względny można oszacować:

$$\left|\frac{m - m_t}{m}\right| \leqslant \frac{2^{-(t+1)}}{1/2} = 2^{-t}$$

Nadmiar i niedomiar

- Występuje, gdy ilość bitów potrzebna do reprezentacji cechy danej liczby jest za mała.
- Nadmiar (overflow) występuje jeśli wartość bezwzględna liczby jest za duża, aby ją reprezentować (nie można już zwiększać cechy, bo brakuje bitów)
- Niedomiar (underflow) występuje jeśli wartość bezwzględna liczby jest za mała, aby ją reprezentować (nie można już zmniejszać cechy, bo brakuje bitów)

Błędy obcięcia (truncation errors)

- Występują, gdy powinnismy wykonać nieskończony ciąg obliczeń
- W praktyce taki ciąg musi być skończony (obcięty)
- Przykłady: ograniczenie szeregu nieskończonego do skończonej liczby składników, aproksymacja pochodnej za pomocą ilorazu różnicowego itp.

Standaryzacja:

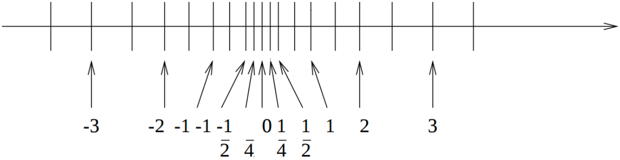
- Reprezentacja zmiennoprzecinkowa została ustandaryzowana w celu ujednolicenia obliczeń
- IEEE754 - 1985; IEEE754 - 2008
- treść standardu IEEE754-2008: `www.dsc.ufcg.edu.br/~cnum/modulos/Modulo2/IEEE754_2008.pdf`
- wywiad z jednym z twórców standardu IEEE754-1985 (William Kahan): `www.goo.gl/AeNLrd`
- kalkulator `https://babbage.cs.qc.cuny.edu/IEEE-754/`

Istotne pojęcia:

ukryta jedynka, normalizacja mantysy, liczby zdenormalizowane, przesunięcie wykładnika

Operacje zmiennopozycyjne

$x, y \in F$
$x + y \in^{?} F$



Problemy:

$\frac{5}{4} + \frac{3}{8} \notin F$ - ze względu na „gęstość” elementów F.

$\frac{7}{2} + \frac{7}{2} \notin F$ - *overflow (nadmiar)*

Operacje zmiennopozycyjne

Oznaczamy: $\oplus$ - dodawanie zmiennopozycyjne,
$fl(x)$ - reprezentacja zmiennoprzecinkowa

działania arytmetyczne na reprezentacjach liczb rzeczywistych muszą być implementowane tak, jakby działanie było wykonywane dokładnie i tylko wynik reprezentowano w zbiorze liczb maszynowych:

$x \oplus y = fl(x + y)$ dla $x + y$ z zakresu F

Problem: jesli dodajemy liczby odległe od siebie, to ta mniejsza może "zniknąć", mimo, że przed dodawaniem była reprezentowalna

$$a + b = 2^{c_a} \cdot \left(m_a + m_b \cdot 2^{-(c_a - c_b)}\right) \Rightarrow \begin{cases} dla |b| \leqslant 1/2 \cdot 2^{-t} \cdot |a| \\ fl(a + b) = a \end{cases}$$

Operacje zmiennopozycyjne

$x \cdot y$ rzadko $\in F$, bo:

- $x \cdot y$ ma $2 \cdot t$ lub $2 \cdot t - 1$ cyfr znaczących,
- *overflow* - bardziej prawdopodobny
- *underflow* - bardziej prawdopodobny
- $\oplus, \odot$
    - są *przemienne*
    - nie są *łączne*, *rozdzielne*

Operacje zmiennopozycyjne

Ogólnie:

$$fl(a\Box b) = \overbrace{rd(a\Box b)}^{\text{inne oznaczenie}} = (a\Box b)\cdot(1+\varepsilon)$$

$$\varepsilon = \varepsilon(a,b,\Box), \varepsilon \leqslant \beta^{1-t}$$

$$\Box = +, -, \cdot, /$$

Definicja

Maszynowe ε - najmniejsza liczba zmiennoprzecinkowa, dla której jeszcze:

$$1 \oplus \varepsilon > 1$$

Wartość maszynowego ϵ określa precyzję obliczeń numerycznych wykonywanych na liczbach zmiennoprzecinkowych.

Zwykle wystarcza znajomość $\varepsilon' = 2^k \cdot \varepsilon, k \approx 1, 2, 3, \ldots$

Zadanie, algorytm

Aby badać wpływ zaburzeń danych na wynik, przyjmujemy pewne założenia (prawdziwe dla wiekszości zadań numerycznych):

Definicja

Zadanie
dla danych $\vec{d} = (d_1, d_2, ..., d_n) \in R_d$
znaleźć wynik $\vec{w} = (w_1, w_2, ..., w_m) \in R_w$ taki, że $\vec{w} = \varphi(\vec{d})$
R_d, R_w - skończenie wymiarowe, unormowane przestrzenie kartezjańskie
$\varphi : D_0 \subset R_d \rightarrow R_w$ - odwzorowanie ciagłe

Definicja

Algorytm A w klasie zadań $\{\varphi, D\}$ jest to sposób wyznaczenia wyniku $\vec{w} = \varphi(\vec{d})$ dla $d \in D \subset D_0$, z dokładną realizacją działań, tj. w zwykłej arytmetyce

Realizacja algorytmu w arytmetyce float - $fl(A(\vec{d}))$

Zastąpienie:

- $d, x, ..$
- arytmetyki

Przez:

- $rd(d), rd(x), ..$
- arytmetykę float

Założenia o reprezentacji danych i wyników

W realizacji w arytmetyce float oczekujemy, że $\vec{d}$ oraz $\vec{w}$ będą reprezentowane z małymi błędami

Założenia

$$||\vec{d} - rd(\vec{d})|| \leqslant \varrho_d ||\vec{d}||$$

$$||\vec{w} - rd(\vec{w})|| \leqslant \varrho_w ||\vec{w}||$$

$$\varrho_d, \varrho_w = \underbrace{k}_{\text{małe, } \approx 10} \cdot \beta^{1-t}$$

Uwarunkowanie zadania

Przyczyna: zamiast $d_i \longrightarrow rd(d_i) = d_i \cdot (1 + \varepsilon)$, $||\varepsilon|| \leqslant \beta^{1-t}$

Definicja

Uwarunkowanie zadania - czułość na zaburzenie danych,
Wskaźniki uwarunkowania zadania - wielkości charakteryzujące wpływ zaburzeń danych zadania na zaburzenie jego rozwiązania.

Definicja

Zadanie nazywamy źle uwarunkowanym, jeżeli niewielkie względne zmiany danych zadania powodują duże względne zmiany jego rozwiązania.

Wskaznik uwarunkowania zadania

- oznaczamy $cond(\varphi(x))$
- znaczenie: jeśli dane znamy z błędem względnym nie większym niż ϵ to błąd względny wyniku obliczenia nie jest większy niż $\epsilon \cdot cond(\varphi(x))$
- np. jeśli $cond(\varphi(x)) = 100$, a dane reprezentowane są z błedem $2^{-23} \approx 10^{-7}$ to błąd względny wyniku jest nie większy niż $10^{-7} \cdot 100 = 10^{-5}$ (pięć cyfr wyniku jest dokładnych)

Uwarunkowanie zadania

Przykład

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \cdot y_i \neq 0 \qquad \begin{array}{l} x_i \to x_i \cdot (1+\alpha_i) \\ y_i \to y_i \cdot (1+\beta_i) \end{array}$$

$$\underbrace{\left| \frac{\sum_{i=1}^{n} x_i \cdot (1+\alpha_i) \cdot y_i \cdot (1+\beta_i) - \sum_{i=1}^{n} x_i \cdot y_i}{\sum_{i=1}^{n} x_i \cdot y_i} \right|}_{\text{błąd względny}} \approx$$

$$\approx \left| \frac{\sum_{i=1}^{n} \cdot x_i \cdot y_i \cdot (\alpha_i + \beta_i)}{\sum_{i=1}^{n} x_i \cdot y_i} \right| \leqslant \max |\alpha_i + \beta_i| \cdot \underbrace{\frac{\sum_{i=1}^{n} |x_i \cdot y_i|}{\left|\sum_{i=1}^{n} x_i \cdot y_i\right|}}_{cond(\vec{x} \cdot \vec{y})}$$

gdy wszystkie x_i, y_i tego samego znaku $\Rightarrow cond(\vec{x} \cdot \vec{y}) = 1$

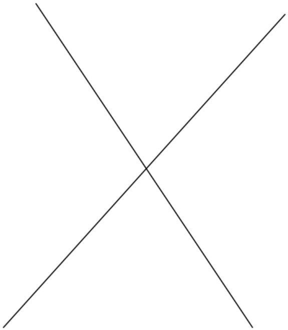
Uwarunkowanie zadania

Poprawa:

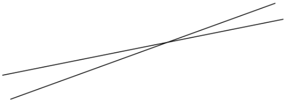
- silniejsza arytmetyka,
- użycie zadania równoważnego

Jakościowo

Układ dwóch równań graficznie:
niezależnie od jakości ołówka (algorytm) - lewa strona - dokładniej.



well conditioned



ill conditioned

Ilościowo

Zadanie: wyznaczenie $f(x)$, przy założeniu: x^* - blisko x

Współczynnik uwarunkowania:

$$\text{ogólnie:} \quad cond(f(x)) = \lim_{x^* \to x} \frac{\left| \frac{f(x) - f(x^*)}{f(x)} \right|}{\left| \frac{x - x^*}{x} \right|} = \left| \frac{x \cdot f'(x)}{f(x)} \right|$$

$$f(x) = \sqrt{x}: \quad cond(f(x)) = \left| \frac{x \cdot \frac{1}{2\sqrt{x}}}{\sqrt{x}} \right| = \frac{1}{2}$$

$$f(x) = \frac{1}{1-x}: \quad cond(f(x)) = \frac{\left| x \cdot \frac{1}{(1-x)^2} \right|}{\left| \frac{1}{1-x} \right|} = \frac{x}{1-x}$$

$$x = (1 + 10^{-6}) \Rightarrow cond(f(x)) \approx 10^6 \qquad (!)$$

Uwarunkowanie zadania - przykład

Przykład 1

$$(x-2)^2 = 10^{-6} \Rightarrow x_{1,\,2} = 2 \mp 10^{-3}$$

ALE: zmiana stałej o $10^{-6} \rightarrow$ to zmiana $x_{1,\,2}$ o 10^{-3}!

Uwarunkowanie zadania - przykład

Przykład 2 - Wilkinson (1963)

Szukamy zer wielomianu:

$p(x) = (x-1)(x-2)\cdot\ldots\cdot(x-19)(x-20) = x^{20} - 210x^{19} + \ldots$

przy zaburzeniu tylko jednego współczynnika (przy x^{19}):

$-210 \rightarrow -210 + 2^{-23}$ $(2^{-23} \approx 1.19\cdot 10^{-7})$

szukamy tak naprawdę zer dla $p(x) + 2^{-23}\cdot x^{19}$

wynik: $\left.\begin{array}{l} 10 \rightarrow 10.095\ldots \mp 0.643\ldots i \\ \ldots \\ 19 \rightarrow 19.502\ldots \mp 1.940\ldots i. \end{array}\right\}$ 10 pierw. zespolonych!

Uwarunkowanie zadania - przykład

Przykład 2 - Wilkinson (1963)

powód: czułość zadania na zaburzenia danych!

$$p(x, \alpha) = x^{20} - \alpha \cdot x^{19} + ... = 0$$

Jak zmienia się miejsce zerowe x_i w stosunku do zmian współczynnika α ?

$$\left.\frac{\partial x}{\partial \alpha}\right|_{x=x_i=i} \leftarrow \text{miara czułości}$$

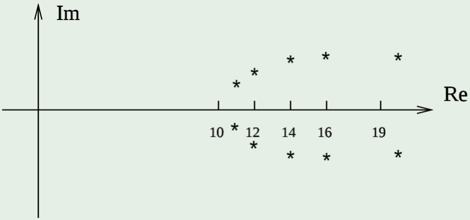
Korzystam z wzoru na pochodną zupełną:

$$\frac{\partial p(x,\alpha)}{\partial x} \cdot \frac{\partial x}{\partial \alpha} + \frac{\partial p(x,\alpha)}{\partial \alpha} = 0$$

$$\frac{\partial x}{\partial \alpha} = -\frac{\frac{\partial p}{\partial \alpha}}{\frac{\partial p}{\partial x}}$$

Uwarunkowanie zadania - przykład

Przykład 2 - Wilkinson (1963)



i	$\left.\frac{\partial x}{\partial \alpha}\right\|_i$
1	$-8.2 \cdot 10^{-18}$
2	$8.2 \cdot 10^{-11}$
5	$-6.1 \cdot 10^{-1}$
6	$5.8 \cdot 10^{1}$
8	$6.0 \cdot 10^{4}$
10	$7.6 \cdot 10^{6}$
15	$-2.1 \cdot 10^{9}$
19	$-3.1 \cdot 10^{8}$
20	$4.3 \cdot 10^{7}$

$$\frac{\partial x}{\partial \alpha} = -\frac{\frac{\partial p}{\partial \alpha}}{\frac{\partial p}{\partial x}} = \frac{x^{19}}{\sum_{i=1}^{20} \prod_{j=1, j\neq i}^{20} (x-j)}$$

$$\left.\frac{\partial x}{\partial \alpha}\right|_{x=i} = \frac{i^{19}}{\prod_{j=1, j\neq i}^{20} (i-j)}, i = 1, 2, ..., 20$$

Algorytmy numerycznie poprawne

Algorytmy numerycznie poprawne

Dają rozwiązania będące nieco zaburzonym dokładnym rozwiązaniem zadania o nieco *zaburzonych* danych. Są to algorytmy najwyższej jakości.

Algorytmy numerycznie poprawne - definicje

Definicja

Algorytm A jest **numerycznie poprawny** w klasie zadań $\{\varphi, D\}$, jeżeli istnieją stałe K_d, K_w takie, że:

- $\forall \vec{d} \in D$,
- dla każdej dostatecznie silnej arytmetyki (β^{1-t})

$\exists \tilde{d} \in D_0$ taki, że:

$$||\vec{d} - \tilde{d}|| \leqslant \varrho_d \cdot K_d \cdot ||\vec{d}||$$

$$||\varphi(\tilde{d}) - fl(A(\vec{d}))|| \leqslant \varrho_w \cdot K_w \cdot ||\varphi(\tilde{d})||$$

$\varphi(\tilde{d})$ - dokładne rozwiązanie zadania o zaburzonych danych $\tilde{d}$
K_w, K_d - wskaźniki kumulacji algorytmu A w klasie zadań $\{\varphi, D\}$
K_w, K_d:

- dla dowolnych danych klasy $\{\varphi, D\}$,
- minimalne $\rightarrow$ jakość A.

Algorytmy numerycznie poprawne - definicje

Definicja

Użyteczne algorytmy - gdy wskaźniki kumulacji rzędu liczby działań

Algorytmy numerycznie poprawne - przykład

Przykład 1 - Iloczyn skalarny

Numeryczna poprawność algorytmu $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i \cdot b_i$

```
A(a⃗, b⃗) :
s := 0;
for i:=1 to n do s := s + a_i · b_i;
```

Algorytmy numerycznie poprawne - przykład

Przykład 1 - Iloczyn skalarny

Realizacja algorytmu: $fl(A(\vec{a}, \vec{b}))$:

1. dane - reprezentacje
$$a_i \longrightarrow \hat{a}_i = rd(a_i) = a_i \cdot (1 + \alpha_i)$$
$$b_i \longrightarrow \hat{b}_i = rd(b_i) = b_i \cdot (1 + \beta_i)$$
2. działania - przybliżone, fl
np. $i = 1, 2, 3$:
$$fl(A(\vec{a}, \vec{b})) = \{[\hat{a}_1 \cdot \hat{b}_1 \cdot (1 + \varepsilon_1) + \hat{a}_2 \cdot \hat{b}_2 \cdot (1 + \varepsilon_2)] \cdot (1 + \delta_2) + \hat{a}_3 \cdot \hat{b}_3 \cdot (1 + \varepsilon_3)\} \cdot (1 + \delta_3);$$
$$\delta_1 = 0$$
i ogólnie:
$$fl(A(\vec{a}, \vec{b})) = \sum_{i=1}^{n} a_i \cdot (1+\alpha_i) \cdot b_i \cdot (1+\beta_i) \cdot (1+\varepsilon_i) \cdot \prod_{j=i}^{n} (1+\delta_j)$$

Algorytmy numerycznie poprawne - przykład

Analiza poprawności numerycznej dla przykładu 1

Istnieją takie $\tilde{a}$ i $\tilde{b}$, dla których da się wskazać K_d i K_w. Na przykład:

- dla $\tilde{a}$ takiego że $\tilde{a}_i = a_i \cdot (1 + \alpha_i)$
- dla $\tilde{b}$ takiego że $\tilde{b}_i = b_i \cdot (1 + \beta_i) \cdot (1 + \varepsilon_i) \cdot \prod_{j=i}^{n}(1 + \delta_j)$
- mamy dokładny wynik: $K_w = 0$
- ale dla zaburzonych danych $\tilde{a}$ i $\tilde{b}$ spełniających warunki :
$$||\vec{a} - \tilde{a}|| \leqslant \beta^{1-t} \cdot ||\vec{a}\,|| \qquad (K_{d_1} = 1)$$
$$||\vec{b} - \tilde{b}|| \leqslant (n+1) \cdot \beta^{1-t} \cdot ||\vec{b}|| \qquad (K_{d_2} = n+1)$$

Tutaj skutki błędów zaokrągleń interpretujemy jako skutki takiego zaburzenia danych, że otrzymany wynik jest dla tych zaburzonych danych dokładny.
Małe zaburzenia oznaczają, że algorytm jest poprawny numerycznie.

Stabilnosc Numeryczna

- algorytmy numerycznie poprawne to algorytmy najwyższej jakości
- udowodnienie numerycznej poprawności algorytmu jest często trudne, wymaga znalezienie wskazników kumulacji niezależnych od danych
- można spróbować zbadać stabilność (słabszy warunek)
- stabilność to minimalny wymóg dla algorytmu
- badamy, jak duży byłby błąd wyniku, gdyby dane oraz wynik zostały zaburzone na poziomie reprezentacji, ale same obliczenia byłyby wykonywane dokładnie

Stabilność numeryczna - przykłady

Przykład 1

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

$\beta = 10, t = 5, x = -5.5$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$\begin{aligned} e^{-5.5} = \; & 1.0 \\ & -5.5 \\ & +15.125 \\ & -27.730 \\ & +38.129 \\ & -41.942 \\ & +38.446 \\ & -30.208 \\ & +20.768 \\ & -12.692 \\ & +6.9803 \\ & -3.4902 \\ & +1.5997 \\ & \ldots \\ \hline & 0.0026363 \end{aligned}$$

ALE $e^{-5.5} = 0.00408677$!

25 składników i brak zmian w sumie

Stabilność numeryczna - przykłady

Przykład 1

Przyczyna: *catastrophic cancellation*

- odejmowanie bliskich liczb - wynikiem jest mała liczba, mająca dużo zer tam, gdzie jej "poprzednicy" mieli cyfry znaczące,
- taka liczba jest normalizowana, zera zapisywane są w wykładniku (cecha), a cała zawartość mantysy przesuwana jest "w lewo"
- nie wiadomo czym zapełnić pojawiające się miejsca w mantysie po prawej stronie (zera lub przypadkowe wartości)

Po zmianie algorytmu: (β, t - j.w.)

$$e^{-5.5} = \frac{1}{e^{5.5}} = \frac{1}{1 + 5.5 + 15.125 + \ldots} = \underbrace{0.0040865}_{0.007\%!}$$

Stabilność numeryczna - przykłady

Przykład 2

$$E_n = \int_0^1 x^n \cdot e^{x-1} dx, \qquad n = 1, 2, \ldots$$

całkowanie przez części:

$$\int_0^1 x^n \cdot e^{x-1} dx = x^n \cdot e^{x-1}\Big|_0^1 - \int_0^1 n \cdot x^{n-1} \cdot e^{x-1} dx$$

$$\Rightarrow \begin{cases} E_n = 1 - n \cdot E_{n-1}, & n = 2, 3, \ldots \\ E_1 = \frac{1}{e} \end{cases}$$

$\beta = 10, t = 6$

$$E_1 \approx 0.367879, \varepsilon \approx 4.412 \cdot 10^{-7}$$

$$E_2 \approx 0.264242$$

$$\ldots$$

$$E_8 \approx 0.118720$$

$$E_9 \approx -0.0684800 \quad !! \quad \text{Bo: } x^9 \cdot e^{x-1} \geqslant 0, x \in [0, 1]$$

Stabilność numeryczna - przykłady

Przykład 2

Powód $\rightarrow \varepsilon(E_1)$:

$$\text{w } E_2 \rightarrow \varepsilon \cdot 2$$

$$\text{w } E_3 \rightarrow \varepsilon \cdot 2 \cdot 3$$

$$\ldots$$

$$\text{w } E_9 \rightarrow \varepsilon \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot 9 = \varepsilon \cdot 9! = \varepsilon \cdot 362880 \approx 0.1601$$

i nawet: $-0.06848 + 0.1601 = 0.0916$ - poprawny!

Stabilność numeryczna - przykłady

Przykład 2

Jak wybrać dobry (stabilny) algorytm?

$$E_{n-1} = \frac{1 - E_n}{n}, \quad n = ..., 3, 2.$$

Na każdym etapie zmniejszamy błąd n razy - **algorytm stabilny**.

$$E_n = \int_0^1 x^n \cdot e^{x-1} dx \leqslant \int_0^1 x^n dx = \frac{x^{n+1}}{n+1}\bigg|_0^1 = \frac{1}{n+1}$$

$$\lim_{n\to\infty} E_n = 0$$

$$E_{20} \approx 0.0 \quad \text{- błąd} \quad \frac{1}{21}$$

$$E_{19} \rightarrow \frac{1}{20} \cdot \frac{1}{21} \approx 0.0024$$

$$\dots$$

$E_{15} \approx 0.0590176 \rightarrow$ wartość dokładna na 6 miejscach znaczących.

Stabilność numeryczna - przykłady

Przykład 2 - definicja

Metoda numerycznie jest **stabilna**, jeżeli mały błąd na dowolnym etapie przenosi się dalej z *malejącą amplitudą*.

$$\varepsilon^{n+1} = g \cdot \varepsilon^n \quad \Rightarrow \text{stabilna: } \left|\varepsilon^{n+1}\right| \leqslant \left|\varepsilon^n\right|, \quad g \text{ - wsp. wzmocnienia}$$

Uwaga:
Jakość wyniku poprawia się wraz z każdym kolejnym etapem obliczeń.

Optymalny poziom błędu rozwiązania

- Dokładne rozwiązanie dla dokładnych danych (idealne) $\vec{w} = \varphi(\vec{d})$
- Dane zaburzone na poziomie reprezentacji $\hat{d} : ||\vec{d} - \hat{d}|| \leqslant \varrho_d ||\vec{d}||$
- Dokładne rozwiązanie dla zaburzonych danych $\hat{w} = \varphi(\hat{d})$
- Zaburzone (na poziomie reprezentacji) rozwiązanie dla zaburzonych danych $\tilde{w} : ||\hat{w} - \tilde{w}|| \leqslant \varrho_w ||\hat{w}||$

Szacujemy różnicę pomiędzy idealnym $\varphi(\vec{d})$, a zaburzonym $\tilde{w}$

$$||\vec{w} - \tilde{w}|| \leqslant ||\vec{w} - \hat{w}|| + ||\hat{w} - \tilde{w}|| \leqslant ||\vec{w} - \hat{w}|| + \varrho_w ||\hat{w}|| \leqslant ||\vec{w} - \hat{w}|| + \varrho_w (||\hat{w} - \vec{w}|| + ||\vec{w}||) \leqslant (1 + \varrho_w) \cdot \max_{\hat{d}} ||\varphi(\vec{d}) - \varphi(\hat{d})|| + \varrho_w ||\vec{w}||$$

Optymalny poziom błędu rozwiązania

Definiujemy optymalny poziom błędu rozwiązania jako:

$$P(\vec{d}, \varphi) = \varrho_w ||\vec{w}|| + \max_{\hat{d}} ||\varphi(\vec{d}) - \varphi(\hat{d})||$$

Poziom ten wynika wyłącznie z przeniesienia błędu reprezentacji danych na wynik obliczeń

Stabilność numeryczna - definicja

Definicja

Algorytm A nazywamy **numerycznie stabilnym** w klasie $\{\varphi, D\}$, jeżeli istnieje stała K taka, że

- $\forall \vec{d} \in D$,
- dla każdej dostatecznie silnej arytmetyki

zachodzi:

$$||\varphi(\vec{d}) - fl(A(\vec{d}))|| \leqslant \underbrace{K}_{\text{małe}} \cdot P(\vec{d}, \varphi)$$

Algorytm numerycznie stabilny gwarantuje uzyskanie rozwiązania z błędem co najwyżej K razy większym, niż optymalny poziom błędu rozwiązania tego zadania.

Złożoność obliczeniowa

Oprócz jakości algorytmu - ważny jego *koszt* - liczba działań arytmetycznych (logicznych) potrzebnych do rozwiązania zadania - algorytmy minimalizujące liczbę działań.

Rezultaty

1. Oszacowanie złożoności obliczeniowej „z dołu”:

Twierdzenie

Jeżeli zadanie ma n danych istotnych, to minimalna liczba działań arytmetycznych potrzebnych do obliczenia wyniku:

$$z(\varphi, D) \geqslant \frac{n}{2}$$

2. Dla wielu zadań można udowodnić, że minimalna liczba działań w algorytmie numerycznie poprawnym musi być istotnie większa od liczby danych.
3. Metody optymalne co do $z(\varphi, D)$ znane są dla niewielu, prostych zadań.

Złożoność obliczeniowa - przykład

Przykład

- prosty
- ilustruje konieczność myślenia *do końca* przy wyborze algorytmu

Zadanie:

$$S = \sum_{i=1}^{n} (-1)^i \cdot i;$$

Złożoność obliczeniowa - przykład

Przykład

A1:

```
s = 0
do 1 i=1, n
      s = s+(-1)**i*i
continue
```

A2:

```
s = 0
do 1 i=1,n,2
      s = s-1
continue
do 1 i=1,n,2
   s = s+1
continue
```

Złożoność obliczeniowa - przykład

Przykład

A3:

```
s = n/2
if (mod(n,2).eq.1) s=-s
```

- ilość operacji nie zależy od n
- nie ma akumulacji błędów
- nie grozi overflow