Metody numeryczne

Aneta Wróblewska

MFI, UMCS

Program wykładu

1. Dokładność
2. Układy równań liniowych
3. Równania nieliniowe – metody iteracyjne
4. Interpolacja
5. Aproksymacja
6. Różniczkowanie
7. Metody rozwiązywania warunków brzegowych równań różniczkowych zwyczajnych
8. Całkowanie
9. Miejsca zerowe wielomianów
10. Generatory liczb pseudolosowych i metody ich testowania
11. Metoda Monte Carlo
12. Metody geometrii obliczeniowej

Literatura podstawowa:

1. Kordecki W., Selwat K., *Metody numeryczne dla informatyków*, Helion, 2020.
2. Moszyński K., *Metody numeryczne dla informatyków*, Helion, 2020r.
3. Krzyżanowski P., *Metody numeryczne, Wydawnictwo Naukowe PWN*, 2024r.

Literatura uzupełniająca:

4. Mikołajczak P., Ważny M., *Metody numeryczne w C++*, Instytut Informatyki UMCS, 2012.
5. Press W. H., Teukolsky S. A., Vetterling W. T. , Flannery B. P., *Numerical Recipes*, Cambridge University Press, 2007
6. Fortuna Z., Macukow B., Wąsowski J., *Metody numeryczne*, WN-T, Warszawa 1982.
7. Mohamad A. A., Benselama A. M. , *Numerical Methods For Engineers: A Practical Approach*, World Scientific Publishing Co Pte Ltd, 2022

Metody numeryczne

Metody numeryczne są działem matematyki stosowanej. Zajmują się badaniem sposobów umożliwiających rozwiązywanie zadań matematycznych za pomocą działań arytmetycznych.

Dlaczego metody numeryczne?

W informatyce i naukach technicznych pojawiają się problemy postaci:

$$f(x) = 0, \quad Ax = b, \quad \int_a^b f(x)\,dx, \quad y' = f(x, y)$$

W praktyce:

- rozwiązanie analityczne często nie istnieje,
- jest zbyt kosztowne obliczeniowo,
- dane są obarczone błędami,
- komputer operuje na skończonej precyzji.

Cel metod numerycznych:

- obliczyć przybliżenie rozwiązania,
- oszacować błąd,
- kontrolować stabilność obliczeń.

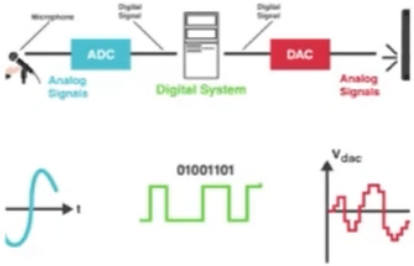
Metody numeryczne

- wykorzystywane środki: analiza matematyczna, algebra (dowody matematyczne uzasadniają poprawność metod),
- nowoczesne metody numeryczne: znajomość języków programowania i bibliotek,
- jedno z podstawowych narzędzi pracy inżyniera,
- służą do analiz i symulacji problemów z fizyki, mechaniki, elektrotechniki, medycyny, ekonomii i wielu innych,
- praktyczne zastosowanie: wykonanie algorytmu jako skończonej liczby działań arytmetycznych oraz logicznych,
- zwykle otrzymujemy rozwiązanie przybliżone wraz z kontrolą błędu.

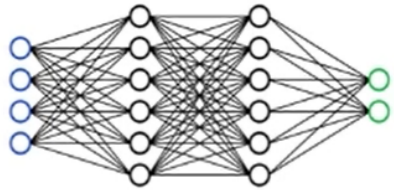
Zastosowania metod numerycznych

Numerical Modeling and Simulation

Signal Processing



Machine Learning

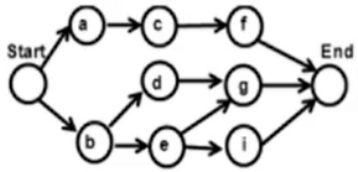


Numerical Methods

Financial Engineering

Cryptography

Operational Research and Optimization



Narzędzia (materiały dodatkowe)

Lista narzędzi wykorzystywanych w analizie numerycznej:

▸ Link

Porównanie narzędzi:

▸ Link

Najważniejsze pojęcia

Dokładność – dopuszczalny błąd wyniku (w praktyce dobierany w zależności od problemu).

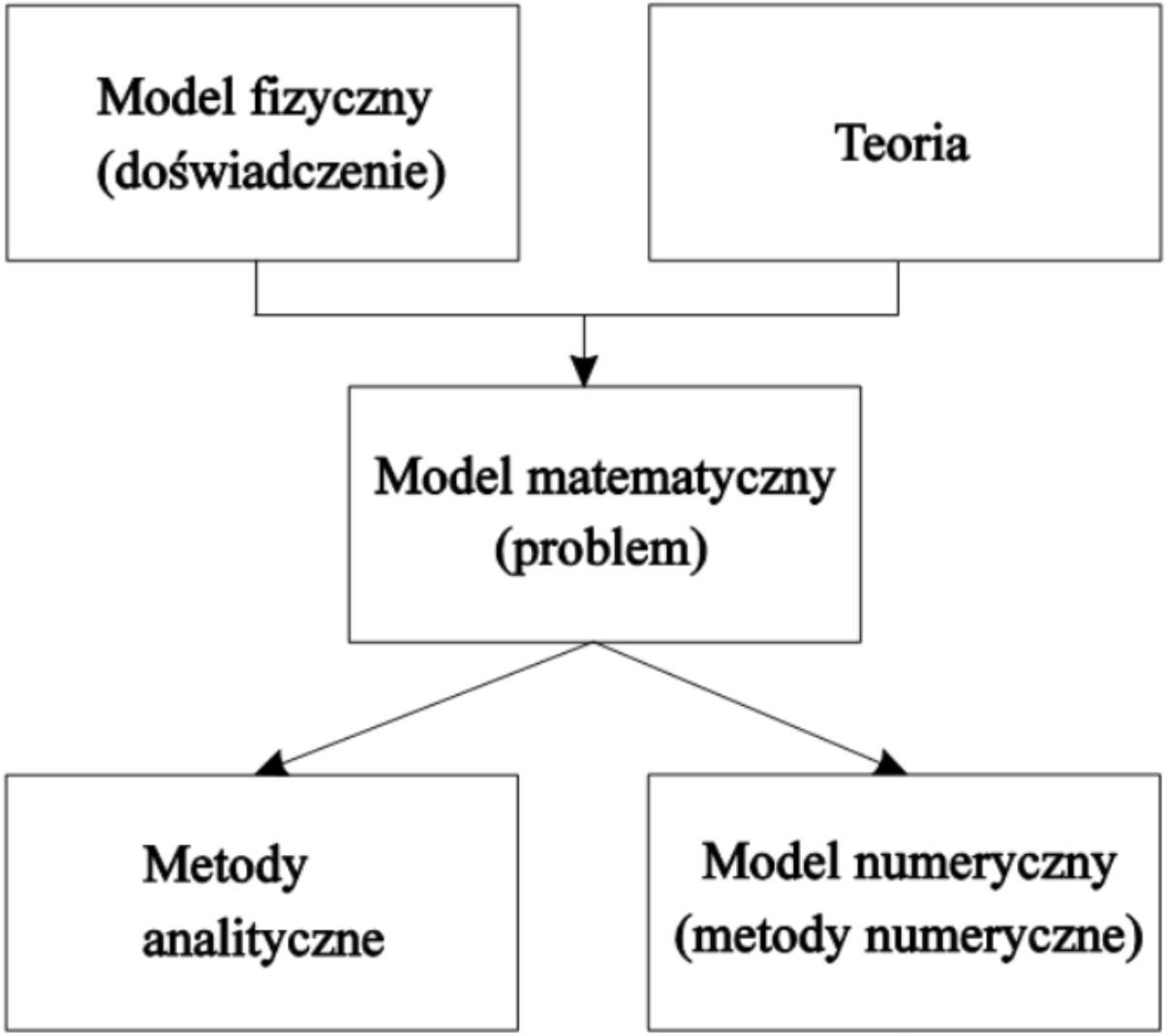
Zbieżność iteracji – stopniowe zbliżanie się do wartości poszukiwanej; kończymy iteracje, gdy osiągniemy zadaną dokładność.

Dyskretyzacja – zastąpienie obiektu ciągłego obiektem o skończonej liczbie węzłów.

Najważniejsze pojęcia cd

Model – celowo uproszczona reprezentacja rzeczywistości.

Model matematyczny (problem) – reprezentacja fragmentu rzeczywistości za pomocą symboli i operatorów matematycznych, z interpretacją odnoszącą się do zjawiska.


Model fizyczny
(doświadczenie)
Teoria
Model matematyczny
(problem)
Metody
analityczne
Model numeryczny
(metody numeryczne)

Wykład nr 1
Dokładność

Plan

1. Miary błędu: bezwzględny, względny, cyfry znaczące
2. Liczby maszynowe: float/double, IEEE 754, epsilon
3. Porównywanie liczb, Inf/NaN, utrata cyfr znaczących
4. Wpływ kolejności działań: gubienie składnika, sumowanie, Kahan
5. Uwarunkowanie i stabilność algorytmów (przykład $a^2 - b^2$)

Błąd bezwzględny

Niech x będzie wartością dokładną, a $\bar{x}$ jej przybliżeniem.

Błąd bezwzględny:

$$\Delta x = |x - \bar{x}|.$$

Jeśli x nie jest znane, zamiast błędu obliczamy jego oszacowanie (kres górny).

Błąd bezwzględny zależy od skali liczby.

Błąd względny

Dla $x \neq 0$:

$$\delta x = \frac{|x - \bar{x}|}{|x|}.$$

Błąd względny mierzy dokładność **niezależnie od rzędu wielkości liczby**.

Przykład: $\Delta x = 0.01$

- dla $x = 1 \rightarrow \delta x = 1\%$
- dla $x = 1000 \rightarrow \delta x = 0.001\%$

W analizie numerycznej kluczowy jest błąd względny.

Źródła błędów w obliczeniach numerycznych

- błędy zaokrągleń wynikające z arytmetyki komputera,
- błędy obcięcia (np. zatrzymanie procesu nieskończonego),
- błędy programisty,
- błędy danych wejściowych (pomiary, zaokrąglenia, stałe fizyczne),
- błędy modelu (uproszczenia modelu matematycznego),
- błędy metody (np. duże błędy dla pewnych danych).

Błędy zaokrągleń

Dokładnie można reprezentować w komputerze tylko:

- liczby całkowite (z pewnego zakresu),
- oraz liczby wymierne o skończonym rozwinięciu binarnym (z pewnego zakresu).

Wiele ułamków dziesiętnych nie ma skończonego rozwinięcia binarnego:

$$\frac{1}{5} = 0.(0011)_2, \qquad \frac{1}{10} = 0.0(0011)_2.$$

Komputer przechowuje przybliżenia obarczone błędem zaokrąglenia.

Cyfry znaczące

Niech $x \neq 0$ ma postać:

$$x = \pm 0.d_1 d_2 d_3 \cdots \times 10^k, \quad d_1 \neq 0.$$

Przybliżenie $\bar{x}$ ma p cyfr znaczących, jeśli:

$$\frac{|x - \bar{x}|}{|x|} \leqslant \frac{1}{2} \cdot 10^{-p}.$$

Liczba cyfr znaczących jest miarą **błędu względnego**.

Cyfry znaczące – interpretacja

Jeśli $\delta x \approx 10^{-p}$, to wynik ma około p cyfr znaczących.

Przykład: $\delta x \approx 10^{-6}$ oznacza około 6 poprawnych cyfr znaczących.

W obliczeniach maszynowych liczba cyfr znaczących jest ograniczona długością mantysy.

Typy całkowite

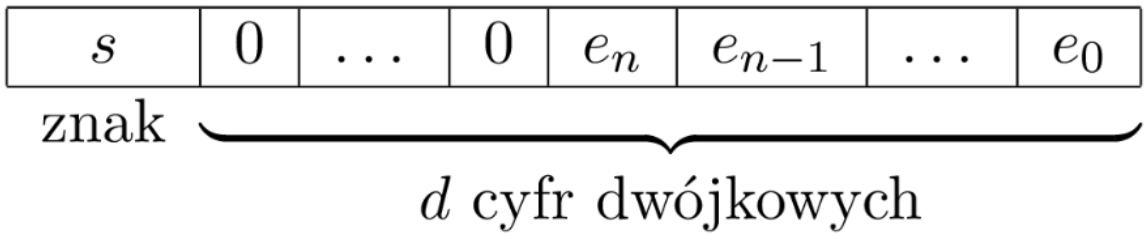
Dowolną liczbę całkowitą x można przedstawić w systemie dwójkowym:

$$x = s \sum_{i=0}^{n} e_i 2^i,$$

gdzie $s \in \{-1, 1\}$ i $e_i \in \{0, 1\}$.

Obliczenia na liczbach całkowitych są zwykle dokładne (do momentu przepełnienia).

Reprezentacja liczby całkowitej i operacje



Zakres: $[-2^d, 2^d - 1]$.

Wyjątki od „dokładności”:

- dzielenie całkowitoliczbowe,
- przepełnienie zakresu (zachowanie zależne od języka/środowiska).

Typy zmiennopozycyjne

Liczba zmiennopozycyjna:

$$x = m \cdot p^c,$$

gdzie m – mantysa, c – wykładnik, p – podstawa (zwykle 2).

W komputerze liczba rzeczywista jest kodowana przez mantysę oraz wykładnik.

Reprezentacja liczby zmiennopozycyjnej w komputerze

Zamiast nieskończonego rozwinięcia mantysy:

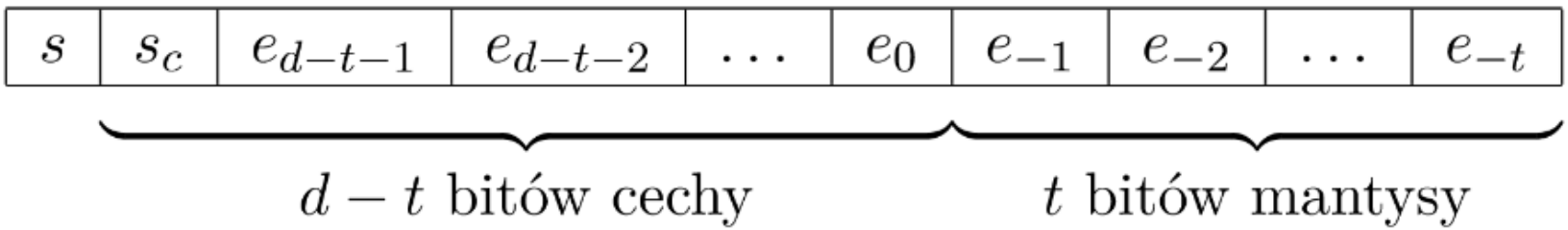
$$m = \sum_{i=1}^{\infty} e_{-i} 2^{-i},$$

używamy przybliżenia obciętego do t bitów:

$$m_t = \sum_{i=1}^{t} e_{-i} 2^{-i}, \quad |m - m_t| \leqslant \frac{1}{2} 2^{-t}.$$

Reprezentacja liczby zmiennopozycyjnej w komputerze

$$x = (-1)^s \cdot 2^c \cdot m_t$$



float vs double w praktyce

Typ	digits10	max_digits10	epsilon	zakres (min..max)
float	6	9	$\approx 1.19 \cdot 10^{-7}$	$\approx 10^{-38}..10^{38}$
double	15	17	$\approx 2.22 \cdot 10^{-16}$	$\approx 10^{-308}..10^{308}$

Wskazówka:

- `setprecision(20)` dla `float` nie zwiększa dokładności — drukuje tylko więcej cyfr.
- Do „wiernego” wypisywania używa się zwykle `max_digits10`.

Standard IEEE 754

Type	Total Size in Bits	Mantissa Size	Exponent Size	Exponent offset
Single Precision	32Bits = 4 Bytes	23 bits	8 Bits	$2^7 - 1 = 127$
Double Precision	64Bits = 8 Bytes	52 bits	11 Bits	$2^{10} - 1 = 1023$

Reprezentacja zmiennopozycyjna i błąd względny

Niech $\mathrm{rd}(x)$ oznacza maszynową reprezentację liczby x (zaokrąglenie do najbliższej liczby maszynowej).

Dla $x \neq 0$ zachodzi model:

$$\mathrm{rd}(x) = x(1+\epsilon), \quad |\epsilon| \leqslant u,$$

gdzie u to **jednostka zaokrąglenia** (unit roundoff).

W arytmetyce binarnej:

$$u = \frac{1}{2} \cdot 2^{-p},$$

gdzie p jest liczbą bitów mantysy.

Mantysa decyduje o dokładności, a wykładnik — o zakresie.

Maszynowy epsilon

$$\varepsilon_{mach} = \min\{\varepsilon > 0 : \ 1 + \varepsilon > 1\}.$$

W praktyce (dla IEEE 754, zaokrąglenie do najbliższej):

$$\varepsilon_{mach} \approx \frac{1}{2} \cdot 2^{-p} = u.$$

Typowo:

- float: $\approx 10^{-7}$
- double: $\approx 10^{-16}$

Model arytmetyki zmiennopozycyjnej

Zakładamy model:

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \epsilon), \quad |\epsilon| \leqslant \varepsilon_{mach},$$

gdzie $\circ \in \{+, -, \cdot, /\}$.

Każda operacja wprowadza błąd względny rzędu:

$$\mathcal{O}(\varepsilon_{mach}).$$

Błędy mogą się kumulować w dłuższych obliczeniach.

Porównywanie liczb: tolerancja absolutna i względna

Porównywanie przez == jest zwykle błędem dla `float/double`.

Stosujemy test mieszany:

$$|a - b| \leqslant \varepsilon \cdot \max(1,\ |a|,\ |b|).$$

Przykład: $\varepsilon = 10^{-12}$

```
double eps = 1e-12;
bool equal = std::fabs(a-b) <= eps * std::max(1.0,
                 std::max(std::fabs(a), std::fabs(b)));
```

Dla wartości bliskich zera dominuje część absolutna, dla dużych — względna.

Dlaczego $0.1 + 0.2 \neq 0.3$?

Liczba 0.1 nie ma skończonej reprezentacji binarnej.

Wynik może wyglądać tak:

$$0.1 + 0.2 = 0.30000000000000004$$

Nie porównujemy:

```
if (a == b)
```

lecz stosujemy tolerancję.

Gubienie małego składnika

$$(10^{20} + 10^{-10}) - 10^{20}$$

W arytmetyce rzeczywistej wynik to 10^{-10}.

W arytmetyce zmiennoprzecinkowej często:

$$\mathrm{fl}(10^{20} + 10^{-10}) = 10^{20},$$

bo 10^{-10} jest mniejsze niż odstęp między liczbami maszynowymi w okolicy 10^{20}.

Wniosek: skala i kolejność działań wpływa na wynik.

Inf / NaN / overflow / underflow

IEEE 754 przewiduje wartości specjalne:

$$1.0/0.0 \to \infty, \qquad 0.0/0.0 \to \mathrm{NaN}.$$

W C++:

```
double x = 1.0/0.0;  // inf
double y = 0.0/0.0;  // nan
std::isinf(x), std::isnan(y)
```

overflow: wynik poza zakresem $\to \pm\infty$
underflow: bardzo małe liczby $\to$ 0 lub liczby zdenormalizowane

Utrata cyfr znaczących

$$f(x) = \sqrt{x^2 + 1} - 1$$

Dla małych x: $\sqrt{x^2 + 1} \approx 1$, więc odejmujemy liczby prawie równe.

Racjonalizacja:

$$f(x) = \frac{x^2}{\sqrt{x^2 + 1} + 1}.$$

Druga postać jest numerycznie stabilniejsza.

Propagacja błędu

Dla funkcji jednej zmiennej (aproksymacja liniowa):

$$\Delta f \approx |f'(x)|\Delta x.$$

Małe błędy danych mogą zostać:

- wzmocnione,
- osłabione,
- zachowane.

Stabilność i niestabilność numeryczna

Niestabilność numeryczna występuje, gdy błędy zaokrągleń i/lub błędy danych są wzmacniane w trakcie obliczeń.

Źródła niestabilności:

- odejmowanie liczb bliskich (utrata cyfr znaczących),
- niekorzystna kolejność działań,
- źle dobrany wzór/postać obliczeń,
- kumulacja błędów w długich obliczeniach.

Stabilność algorytmu

Algorytm jest stabilny, jeśli nie wzmacnia nieuniknionych błędów zaokrągleń.

Jeżeli błąd końcowy jest rzędu:

$$\mathcal{O}(\varepsilon_{mach}),$$

to algorytm jest stabilny.

Jeśli błędy rosną znacząco bardziej — algorytm jest niestabilny.

Uwarunkowanie zadania numerycznego

Uwarunkowanie określa wrażliwość rozwiązania na zaburzenia danych wejściowych.

Zadanie jest **źle uwarunkowane**, gdy nawet niewielkie błędy danych mogą powodować duże błędy wyniku **niezależnie od algorytmu**.

Liczba uwarunkowania (wersja względna)

Propagacja błędu (aproksymacja liniowa):

$$\Delta f \approx f'(x)\,\Delta x$$

Wersja względna:

$$\frac{|\Delta f|}{|f(x)|} \approx \underbrace{\left|\frac{xf'(x)}{f(x)}\right|}_{\kappa(x)} \cdot \frac{|\Delta x|}{|x|}$$

Definicja:

$$\boxed{\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right|}$$

Interpretacja:

$$\text{względny błąd wyjścia} \approx \kappa(x) \cdot \text{względny błąd wejścia}$$

Źle uwarunkowany problem:

Jeżeli

$$\kappa(x) \gg 1,$$

to mały błąd danych może powodować duży błąd wyniku.

Uwarunkowanie jest własnością problemu (funkcji), a nie algorytmu.

Przykłady liczby uwarunkowania

Przykład 1: $f(x) = x^2$

$$f'(x) = 2x$$

$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x \cdot 2x}{x^2}\right| = 2$$

Wniosek: względny błąd wyniku jest około 2 razy większy niż względny błąd wejścia. Problem dobrze uwarunkowany.

Przykłady liczby uwarunkowania

Przykład 2: $f(x) = \sin x$

$$f'(x) = \cos x$$

$$\kappa(x) = \left| \frac{x \cos x}{\sin x} \right|$$

Dla małych x:

$$\sin x \approx x, \quad \cos x \approx 1$$

$$\kappa(x) \approx \left| \frac{x}{x} \right| = 1$$

Wniosek: w pobliżu zera problem jest dobrze uwarunkowany.

Uwaga: Gdy $f(x)$ jest bardzo małe (np. blisko miejsca zerowego), $\kappa(x)$ może być bardzo duże — problem staje się źle uwarunkowany.

Uwarunkowanie vs stabilność

Uwarunkowanie – własność problemu $w = \Phi(d)$.

Stabilność – własność algorytmu (sposobu obliczania).

Cel: stabilny algorytm dla możliwie dobrze uwarunkowanego problemu.

Relacja między pojęciami

- dobre uwarunkowanie + stabilny algorytm $\rightarrow$ wynik dokładny,
- dobre uwarunkowanie + niestabilny algorytm $\rightarrow$ duży błąd,
- złe uwarunkowanie + stabilny algorytm $\rightarrow$ błąd „z natury problemu”,
- złe uwarunkowanie + niestabilny algorytm $\rightarrow$ katastrofa numeryczna.

Przykład: $a^2 - b^2$

$$\Phi(a, b) = a^2 - b^2$$

Dwa algorytmy:

- A1: $a \cdot a - b \cdot b$
- A2: $(a + b)(a - b)$

Algebraicznie równoważne, numerycznie — niekoniecznie.

Analiza w modelu fl

$$\mathrm{fl}(x \circ y) = (x \circ y)(1+\epsilon), \quad |\epsilon| \leqslant \varepsilon_{mach}.$$

W A1 występuje odejmowanie liczb bliskich, gdy $a \approx b$:

$$a^2 \approx b^2 \quad \Rightarrow \quad a^2 - b^2 \text{ jest małe.}$$

To sprzyja utracie cyfr znaczących i wzmacnianiu błędu względnego.

Dlaczego A1 może być niestabilny?

Jeśli $a \approx b$, to wynik jest małą różnicą dużych liczb.

Wtedy względny błąd może zostać silnie wzmocniony.

W A2:

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

mały czynnik $a - b$ występuje bezpośrednio — zwykle stabilniej.

Wniosek

Równoważność algebraiczna nie oznacza równoważności numerycznej.

A2 bywa stabilniejszy niż A1, gdy $a \approx b$.

To klasyczny przykład **katastrofalnej utraty cyfr znaczących**.

Problem sumowania

Rozważamy ogólną sumę:

$$S_n = \sum_{k=1}^{n} a_k$$

Cel analizy:

- porównać różne algorytmy sumowania,
- zbadać wpływ precyzji (float vs double),
- przeanalizować propagację błędu numerycznego.

Sumowanie jest jedną z najczęstszych operacji w obliczeniach numerycznych.

Dlaczego sumowanie jest podchwytliwe?

Niech a_k będą liczbami zmiennoprzecinkowymi.

Możliwe trudności:

- składniki mają różne rzędy wielkości,
- mogą zmieniać znak,
- mogą być bardzo małe względem aktualnej sumy,
- liczba składników n może być bardzo duża.

Matematycznie suma jest dobrze określona.
Problem leży w arytmetyce zmiennoprzecinkowej.

Klasyczne sumowanie

Algorytm:

$$s = 0, \qquad s \leftarrow s + a_k$$

Model błędu:

$$fl(x + y) = (x + y)(1 + \varepsilon), \qquad |\varepsilon| \leqslant \varepsilon_{mach}$$

Po n krokach:

$$\text{błąd względny} = O(n\,\varepsilon_{mach})$$

Im większe n, tym większa kumulacja błędów.

Algorytm Gilla–Møllera

Idea: przechowywać sumę oraz poprawkę (kompensację).

```
s = 0
p = 0
for a_k:
    t = s + a_k
    p = p + (a_k - (t - s))
    s = t
return s + p
```

Zmniejsza utratę cyfr znaczących, ale błąd nadal rośnie z n.

Algorytm Kahana

Algorytm kompensowanego sumowania:

```
sum = 0
c = 0
for a_k:
    y = a_k - c
    t = sum + y
    c = (t - sum) - y
    sum = t
return sum
```

Kompensuje utratę małych składników.
Błąd nie narasta liniowo z n.

Jakie czynniki możemy badać przy sumowaniu?

- porównanie błędów:
  - klasyczne sumowanie,
  - Gill–Møller,
  - Kahan,
- wpływ liczby składników n,
- wpływ precyzji obliczeń,
- związek z:
  - propagacją błędu,
  - ε_{mach},
  - stabilnością algorytmów.

Podsumowanie wykładu

- Każde obliczenie numeryczne jest przybliżeniem.
- Źródłem błędów jest reprezentacja maszynowa.
- Algorytmy mogą być stabilne lub niestabilne.
- Problemy mogą być dobrze lub źle uwarunkowane.
- Sposób liczenia ma znaczenie.