

Numerička matematika

1. predavanje — dodatak

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja — dodatka

- Prikaz realnih brojeva — “floating-point” standard:
 - Osnovni oblik “floating-point” prikaza — mantisa i eksponent.
 - Greške zaokruživanja u prikazu.
 - Pojam “jedinične greške zaokruživanja”.
 - IEEE standard — tipovi: single, double, extended.
- Greške zaokruživanja u aritmetici realnih brojeva:
 - Greške zaokruživanja osnovnih aritmetičkih operacija.
 - Opasno ili “katastrofalno” kraćenje.
 - “Širenje” grešaka zaokruživanja.
 - Stabilni i nestabilni algoritmi.

Sadržaj predavanja — dodatka

- Primjeri širenja grešaka zaokruživanja i izbjegavanja nestabilnosti:
 - Parcijalne sume harmonijskog reda.
 - Korijeni kvadratne jednadžbe.
 - Transformacije nestabilnih formula.

Prikaz “realnih” brojeva u računalu — IEEE standard

Uvod u prikaz realnih brojeva

Kako pohraniti “jako velike” ili “jako male” brojeve?
Recimo (dekadski pisano):

67800000000.0 0.000002078

Koristimo tzv. **znanstvenu** notaciju u kojoj

- prvo pišemo vodeće značajne znamenke broja,
- a zatim pišemo faktor koji ima oblik baza na odgovarajući eksponent, tj. potenciju baze.

Uz dogovor da vodeći dio bude između 1 i 10 (strogo ispod),
to izgleda ovako:

$6.78 \cdot 10^{10}$ $2.078 \cdot 10^{-6}$.

Prikaz realnih brojeva

U računalu se binarni zapis realnog broja pohranjuje u znanstvenom formatu:

$$\text{broj} = \text{predznak} \cdot \text{mantisa} \cdot 2^{\text{eksponent}}.$$

Mantisa se uobičajeno (postoje iznimke!) pohranjuje u tzv. **normaliziranom** obliku, tj.

$$1 \leq \text{mantisa} < (10)_2.$$

I za pohranu **mantise** i za pohranu **eksponenta** rezervirano je **konačno** mnogo binarnih znamenki. Posljedice:

- prikaziv je samo neki **raspon** realnih brojeva,
- niti svi brojevi unutar prikazivog raspona **nisu prikazivi** (mantisa predugačka) \implies **zaokruživanje**.

Prikaz realnih brojeva (nastavak)

Primjer: Znanstveni prikaz binarnih brojeva:

$$1010.11 = 1.01011 \cdot 2^3$$

$$0.0001011011 = 1.01011 \cdot 2^{-4}$$

Primijetite da se vodeća jedinica u normaliziranom obliku **ne mora** pamtiti (ako je broj $\neq 0$).

- Taj bit se može upotrijebiti za pamćenje dodatne znamenke mantise.

Tada se vodeća jedinica zove **skriveni bit** (engl. hidden bit) — jer se **ne pamti**.

Ipak ovo je samo pojednostavljeni prikaz realnih brojeva.

Stvarni prikaz realnih brojeva

Najznačajnija promjena obzirom na pojednostavljeni prikaz:

- eksponent se prikazuje u “zamaskiranoj” ili “pomaknutoj” formi (engl. “biased form”).

To znači da se stvarnom eksponentu

- dodaje konstanta — takva da je “pomaknuti” eksponent uvijek pozitivan.

Ta konstanta ovisi o broju bitova za eksponent i bira se tako da je prikaziva

- recipročna vrijednost najmanjeg pozitivnog normaliziranog broja.

Takav “pomaknuti” eksponent naziva se karakteristika, a normaliziranu mantisu neki zovu i signifikand.

Oznake

Oznake:

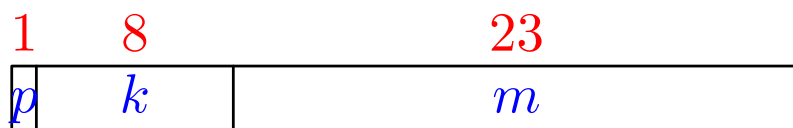
- **Crveno** — duljina odgovarajućeg polja (u bitovima), bitove brojimo od 0 zdesna nalijevo (kao i obično),
- p — predznak: 0 za pozitivan broj, 1 za negativan broj,
- k — karakteristika,
- m — mantisa (signifikand).
- Najznačajniji bit u odgovarajućem polju je najljeviji.
- Najmanje značajan bit u odgovarajućem polju je najdesniji.

Stvarni prikaz tipa single

“Najkraći” realni tip je tzv. realni broj **jednostruke** točnosti — u C-u poznat kao **float**.

On ima sljedeća svojstva:

- duljina: 4 byte-a (32 bita), podijeljen u **tri** polja.



- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- **stvarni eksponent** broja e , $e \in \{-126, \dots, 127\}$,
- **karakteristika** $k = e + 127$, tako da je $k \in \{1, \dots, 254\}$,
- **karakteristike** $k = 0$ i $k = 255$ koriste se za “posebna stanja”.

Stvarni prikaz tipa single (nastavak)

Primjer: Broj $(10.25)_{10}$ prikažite kao broj u jednostrukoj točnosti.

$$\begin{aligned}(10.25)_{10} &= \left(10 + \frac{1}{4}\right)_{10} = (10 + 2^{-2})_{10} \\ &= (1010.01)_2 = 1.01001 \cdot 2^3.\end{aligned}$$

Prema tome je:

$$p = 0$$

$$k = e + 127 = (130)_{10} = (2^7 + 2^1)_{10} = 1000\ 0010$$

$$m = 0100\ 1000\ 0000\ 0000\ 0000\ 000$$

Prikazi nule: $k = 0, m = 0$

Realni broj **nula** ima **dva** prikaza:

● mantisa i karakteristika su joj **nula**,
a predznak može biti

● **0** — “pozitivna nula”, ili

● **1** — “negativna nula”.

Ta dva prikaza nule su:

$$+0 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

$$-0 = 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

Smatra se da su vrijednosti ta dva broja **jednake** (kad se uspoređuju).

Denormalizirani brojevi: $k = 0, m \neq 0$

Ako je $k = 0$, a postoji **barem jedan** znak mantise koji **nije** nula, onda se kao eksponent uzima -126 . Mantisa takvog broja **nije normalizirana** i počinje s $0.m$.

Takvi brojevi zovu se **denormalizirani brojevi**.

Primjer: Kako izgleda prikaz realnog broja

$$0.000\ 0000\ 0000\ 0000\ 0000\ 1011 \cdot 2^{-126}?$$

Rješenje:

$$p = 0$$

$$k = 0000\ 0000$$

$$m = 000\ 0000\ 0000\ 0000\ 0000\ 1011$$

Plus i minus beskonačno: $k = 255, m = 0$

Ako je $k = 255$, a mantisa jednaka 0, onda

• $p = 0$ — prikaz $+\infty$, skraćena oznaka **+Inf**,

• $p = 1$ — prikaz $-\infty$, skraćena oznaka **-Inf**.

Primjer: Prikaz broja $+\infty$ ($-\infty$) je

$$p = 0 \quad (p = 1)$$

$$k = 1111 \ 1111$$

$$m = 000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000$$

Nije broj: $k = 255, m \neq 0$

Ako je $k = 255$ i postoji bar jedan bit mantise različit od nule, onda je to signal da se radi o pogrešci (recimo dijeljenje s nulom, vađenje drugog korijena iz negativnog broja i sl.)

Tada se takva pogreška kodira znakom za Not a Number ili, skraćeno, s NaN:

$$p = 0$$

$$k = 1111 \ 1111$$

$$m = 000 \ 0000 \ 0000 \ 0101 \ 0000 \ 0000$$

Greške zaokruživanja

Postoje realni brojevi koje **ne možemo egzaktno** spremiti u računalo, čak i kad su **unutar** prikazivog raspona brojeva. Takvi brojevi imaju **predugačku mantisu**.

Primjer: Realni broj (u binarnom zapisu)

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

ima **25** znamenki mantise i **ne može** se egzaktno spremiti u realni broj jednostruke preciznosti **float** u **C**-u, koji ima **23 + 1** znamenki za mantisu.

Procesor tada pronalazi dva najbliža **prikaziva** susjeda a_- , a_+ , broju a , takva da vrijedi

$$a_- < a < a_+.$$

Greške zaokruživanja (nastavak)

U našem primjeru je:

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

$$a_- = 1.0001\ 0000\ 1000\ 0011\ 1001\ 011$$

$$a_+ = 1.0001\ 0000\ 1000\ 0011\ 1001\ 100$$

Nakon toga, **zaokružuje** se rezultat. Zaokruživanje može biti:

- 🔴 prema **najbližem** broju (**standardno**, engl. **default**, za IA-32 procesore) — ako su dva susjeda jednako udaljena od a , izabire **parni** od ta dva broja,
- 🔴 prema **dolje**, tj. prema $-\infty$,
- 🔴 prema **gore**, tj. prema ∞ ,
- 🔴 prema **nuli**, tj. odbacivanjem “viška” znamenki.

Greške zaokruživanja (nastavak)

Standardno zaokruživanje u našem primjeru:

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

$$a_- = 1.0001\ 0000\ 1000\ 0011\ 1001\ 011$$

$$a_+ = 1.0001\ 0000\ 1000\ 0011\ 1001\ 100$$

Ovdje su a_- i a_+ jednako udaljeni od a , pa je zaokruženi a jednak a_+ , jer a_+ ima **parni** zadnji bit (jednak je 0).

Jedinična greška zaokruživanja

Dakle, ako je $x \in \mathbb{R}$ unutar raspona brojeva prikazivih u računalu, onda se umjesto x sprema zaokruženi broj prikazivi $fl(x)$.

Time smo napravili **grešku zaokruživanja** $\leq \frac{1}{2}$ “zadnjeg bita” mantise i taj broj se zove

🔴 **jedinična greška zaokruživanja** (engl. unit roundoff).

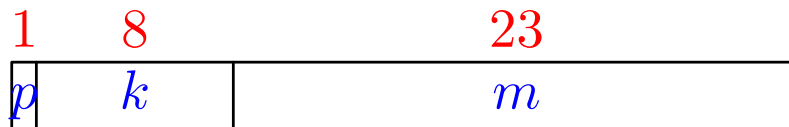
Standardna oznaka je u . Za **float** je $u = 2^{-24} \approx 5.96 \cdot 10^{-8}$.
Vrijedi

$$fl(x) = (1 + \varepsilon)x, \quad |\varepsilon| \leq u,$$

gdje je ε relativna greška napravljena tim zaokruživanjem.
Dakle, imamo **vrlo malu** relativnu grešku.

Prikaz brojeva jednostruke točnosti — sažetak

IEEE tip `single` = `float` u C-u:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-127)} * (1.m) & \text{ako je } 0 < k < 255, \\ (-1)^p * 2^{(-126)} * (0.m) & \text{ako je } k = 0 \text{ i } m \neq 0, \\ (-1)^p * 0 & \text{ako je } k = 0 \text{ i } m = 0, \\ (-1)^p * \text{Inf} & \text{ako je } k = 255 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 255 \text{ i } m \neq 0. \end{cases}$$

Raspon tipa float

Najveći prikazivi pozitivni broj je
 $\text{FLT_MAX} \approx 3.402823466 \cdot 10^{38}$, s prikazom

$$p = 0$$

$$k = 1111\ 1110$$

$$m = 111\ 1111\ 1111\ 1111\ 1111\ 1111$$

Najmanji prikazivi normalizirani pozitivni broj je
 $\text{FLT_MIN} \approx 1.175494351 \cdot 10^{-38}$, s prikazom

$$p = 0$$

$$k = 0000\ 0001$$

$$m = 000\ 0000\ 0000\ 0000\ 0000\ 0000$$

Raspon tipa float

Simboličke konstante `FLT_MAX`, `FLT_MIN` i još poneke vezane uz tip `float`, definirane su u datoteci zaglavlja `float.h` i mogu se koristiti u C programima.

Uočite:

- $1/\text{FLT_MIN}$ je **egzaktno** prikaziv (nađite prikaz),
- $1/\text{FLT_MAX}$ **nije egzaktno** prikaziv i zalazi u denormalizirane brojeve (tzv. “gradual underflow”).

Najmanji prikazivi denormalizirani pozitivni broj je $2^{-126} \cdot 2^{-23} = 2^{-149} \approx 1.4013 \cdot 10^{-45}$, s prikazom

$$p = 0$$

$$k = 0000\ 0000$$

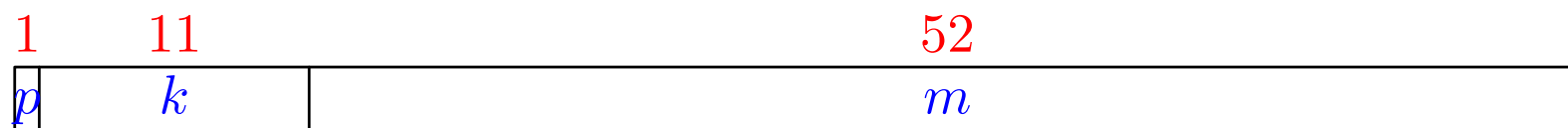
$$m = 000\ 0000\ 0000\ 0000\ 0000\ 0001$$

Stvarni prikaz tipa double

“Srednji” realni tip je tzv. realni broj **dvostruke** točnosti — u C-u poznat kao **double**.

On ima sljedeća svojstva:

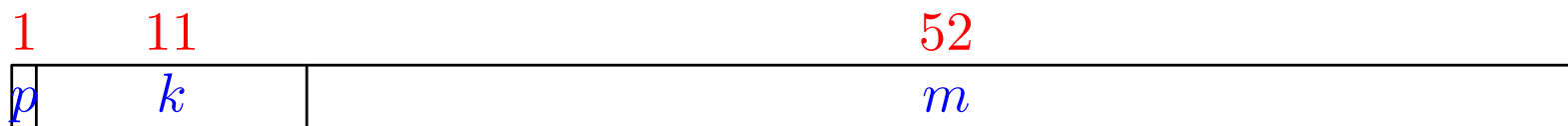
- Duljina: 8 byte-a (64 bita), podijeljen u **tri** polja.



- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- **stvarni eksponent** broja e , $e \in \{-1022, \dots, 1023\}$,
- **karakteristika** $k = e + 1023$, tako da je $k \in \{1, \dots, 2046\}$,
- **karakteristike** $k = 0$ i $k = 2047$ — “posebna stanja”.

Prikaz brojeva dvostruke točnosti — sažetak

IEEE tip `double` = `double` u C-u:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-1023)} * (1.m) & \text{ako je } 0 < k < 2047, \\ (-1)^p * 2^{(-1022)} * (0.m) & \text{ako je } k = 0 \text{ i } m \neq 0, \\ (-1)^p * 0 & \text{ako je } k = 0 \text{ i } m = 0, \\ (-1)^p * \text{Inf} & \text{ako je } k = 2047 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 2047 \text{ i } m \neq 0. \end{cases}$$

Jedinična greška i raspon tipa double

Jedinična greška zaokruživanja za `double` je

$$u = 2^{-53} \approx 1.11 \cdot 10^{-16}.$$

Broj $1 + 2u$ je najmanji prikazivi broj strogo veći od 1. Postoji

$$\text{DBL_EPSILON} = 2u \approx 2.2204460492503131 \cdot 10^{-16}.$$

Najveći prikazivi pozitivni broj je

$$\text{DBL_MAX} \approx 1.7976931348623158 \cdot 10^{308}.$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\text{DBL_MIN} \approx 2.2250738585072014 \cdot 10^{-308}.$$

Tip extended

Stvarno računanje (na IA-32) se obično radi u “proširenoj” točnosti — u C-u možda dohvatljiv kao `long double`.

On ima sljedeća svojstva:

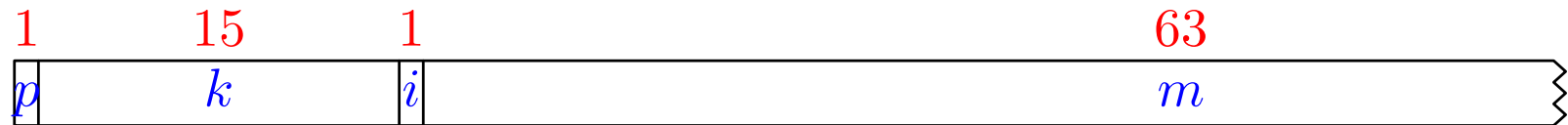
- Duljina: 10 byte-a (80 bita), podijeljen u četiri polja.



- u mantisi se pamti vodeći bit i mantise,
- stvarni eksponent broja e , $e \in \{-16382, \dots, 16383\}$,
- karakteristika $k = e + 16383$, tako da je $k \in \{1, \dots, 32766\}$,
- karakteristike $k = 0$ i $k = 32767$ — “posebna stanja”.

Prikaz brojeva proširene točnosti — sažetak

IEEE tip extended:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-16383)} * (i.m) & \text{ako je } 0 \leq k < 32767, \\ (-1)^p * \text{Inf} & \text{ako je } k = 32767 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 32767 \text{ i } m \neq 0. \end{cases}$$

Uočite da **prva** mogućnost uključuje:

• $+0$, -0 i **denormalizirane** brojeve (za $k = 0$),
jer se **pamti** vodeći “cjelobrojni” bit i mantise.

Zaokruživanje u aritmetici

Osnovna pretpostavka za realnu aritmetiku u računalu:

- za sve četiri osnovne aritmetičke operacije vrijedi ista ocjena greške zaokruživanja kao i za prikaz brojeva.

Isto vrijedi i za neke matematičke funkcije, poput $\sqrt{\quad}$, ali ne vrijedi za sve funkcije (na pr. za \cos i \sin u okolini nule).

Preciznije: Neka \circ označava bilo koju operaciju $+$, $-$, $*$, $/$. Za prikazive brojeve u dozvoljenom rasponu $x, y \in \mathcal{F}$, takve da je i egzaktni rezultat $x \circ y$ u dozvoljenom rasponu (tj. u \mathcal{F}), vrijedi ocjena relativne greške

$$fl(x \circ y) = (1 + \varepsilon)(x \circ y), \quad |\varepsilon| \leq u.$$

Broj ε ovisi o x , y , operaciji \circ i aritmetici računala.

Zaokruživanje u aritmetici (nastavak)

Ova ocjena je **ekvivalentna idealnom** izvođenju aritmetičkih operacija:

- **egzaktno** izračunaj rezultat operacije $x \circ y$,
- **zaokruži** ga, pri spremanju rezultata u memoriju.

To **ne znači** da računalo **zaista** tako i računa. Naime,

- za $+$, $-$, $*$ to bi se još i moglo napraviti (egzaktne rezultati imaju konačan binarni prikaz),
- ali kod **dijeljenja** to sigurno ne ide (egzaktan kvocijent može imati beskonačan binarni prikaz).

Dakle, **važno** je samo da dobijemo istu **ocjenu greške** kao u “idealnom” računanju, a **nije važno** kako se stvarno računa!

Zaokruživanje u aritmetici (nastavak)

U principu, to **dozvoljava** da se rezultati **ponešto razlikuju** na raznim računalima (ovisno o stvarnoj realizaciji aritmetike), ali

- **ocjena relativne greške mora** biti ista.

Uočite da “mjesto za razlike” nema puno, tj. razlike su zaista **rijetke**.

Napomena: oznaka $fl(\text{izraz})$, općenito, označava **izračunatu** vrijednost izraza, tj. ona mora biti **prikaziva**.

- Ali, to **ne znači**: “**izračunaj egzaktno**”, pa “**zaokruži**”,
- nego: **izračunaj sve** operacije i funkcije **u aritmetici računala** (tj. i svaki **međurezultat** mora biti **prikaziv**).

Zaokruživanje u aritmetici (nastavak)

Još nekoliko napomena o točnosti aritmetike. Relacija

$$fl(x \circ y) = (1 + \varepsilon)(x \circ y), \quad |\varepsilon| \leq u,$$

kaže da **izračunati rezultat** $fl(x \circ y)$ ima **malu relativnu grešku** obzirom na egzakti rezultat $x \circ y$.

Obratite pažnju na **uvjete** uz koje vrijedi taj zaključak:

- **ulazne vrijednosti** moraju biti **prikazive** (što je jasno), ali i **u dozvoljenom rasponu**, tj. **normalizirane**,
- **egzakti rezultat** mora, također, biti **u dozvoljenom rasponu**.

Bez **oba** uvjeta, zaključak **ne vrijedi**.

Zaokruživanje u aritmetici (nastavak)

Prvi uvjet — na ulaz, “zvuči razumno” i s njim nije teško izaći na kraj.

Ako je (barem jedna) ulazna vrijednost nula, onda su operacije

- ili egzaktne, tj. nema greške

(operacije $+$, $-$, $*$, $0/y$, uz $y \neq 0$, i funkcija $\sqrt{\quad}$),

- ili dijelimo s nulom, pa dobivamo `Inf`, `NaN` ili grešku.

Za sve ostale moguće nenormalizirane operande i ne očekujemo neki “točan” rezultat.

U svakom slučaju, ulazne vrijednosti uvijek možemo testirati (tj. provjeriti) u programu i tako kontrolirati što se događa.

Zaokruživanje u aritmetici (nastavak)

Drugi uvjet — na rezultat, je mnogo teže kontrolirati:

- izračunati rezultat “vidimo” tek kad ga izračunamo, i tad je sve gotovo, a egzakti rezultat ionako ne znamo.

Općenito, može nam se dogoditi da je egzakti rezultat izvan prikazivog raspona (kao i kod prikaza brojeva):

- “blizu nula” po apsolutnoj vrijednosti — tzv. **underflow**, i
 - računalo tada “šutke” vraća nulu ili nenormalizirani broj “blizu” nule (bez poruke o grešci),
- “prevelik” po apsolutnoj vrijednosti — tzv. **overflow**, i
 - računalo tada vraća **Inf** ili javlja grešku (i prekida izvršavanje programa).

Zaokruživanje u aritmetici (nastavak)

Naravno, **treća** mogućnost je da radimo **nedozvoljenu** operaciju, pa je rezultat **NaN** ili **greška**, ali to se može spriječiti kontrolom unaprijed.

Ni u jednom od ova **tri** slučaja **ne dobivamo malu relativnu grešku**, dakle, ranija ocjena **ne vrijedi**.

Oprez: čisto statistički gledano, po svim dozvoljenim operandima, **množenje** i **dijeljenje** relativno **često** daju (**egzakti**) rezultat **izvan prikazivog raspona** (na pr. x^2).

Iako **egzakti** rezultat operacije **ne znamo** unaprijed, to **ne znači** da

- nema mogućnosti kontrole pojave rezultata **izvan prikazivog raspona** (i pripadnog **gubitka točnosti**).

Zaokruživanje u aritmetici (nastavak)

Naprotiv, gomila nepotrebnih pojava **underflowa** i, posebno, **overflowa**

- može se izbjeći pažljivim projektiranjem algoritama.

Overflow je, općenito, **opasniji**, jer

- nema “gradiranog” prijelaza (kao kod underflowa),
- najčešće (još uvijek) završava greškom, tj. prekidom izvršavanja programa.

Zato se računanje obično “**skalira**” tako da se izbjegne overflow. U većini slučajeva, to se postiže

- jednostavnim transformacijama standardnih formula.**

Primjeri malo kasnije.

Širenje grešaka zaokruživanja

Vidimo da gotovo **svaki** izračunati rezultat ima neku **grešku**.
Osim toga,

- zaokruživanje se vrši nakon **svake pojedine operacije**.

(Najlakše je stvar zamišljati kao da zaokruživanje ide “na kraju” operacije, iako je ono “dio operacije”.)

Kad imamo puno aritmetičkih operacija (inače nam računalo ne treba), dolazi do tzv.

- akumulacije grešaka zaokruživanja**.

Malo pogrešni rezultati (možda već od čitanja), ulaze u operacije, koje opet malo griješe, i tako redom ...

- greške se “šire” kroz sve što računamo!**

Širenje grešaka zaokruživanja (nastavak)

Očekujemo da greške “rastu”, ali koliko?

● “Pomalo”, ili “brzo”?

Ključno pitanje: Možemo li nešto reći o tom “rasprostiranju” grešaka zaokruživanja?

Možemo!

Ali, (uvijek ima neki “ali”),

● put do odgovora nije nimalo jednostavan,

● i treba ga provesti za svaki pojedini proračun posebno.

Gdje je problem?

Širenje grešaka zaokruživanja (nastavak)

Nažalost,

- aritmetika računala nije egzaktna
- i u njoj ne vrijede uobičajeni zakoni za operacije.

Na primjer, za zbrajanje i množenje u računalu (\oplus , \odot),

- nema asocijativnosti,
- nema distributivnosti.

Dakle, poredak izvršavanja operacija je bitan!

Zapravo, jedino standardno pravilo koje vrijedi je

- komutativnost za zbrajanje i za množenje.

(Do nedavno je postojalo računalo na kojem ni to nije vrijedilo).

Primjer neasocijativnosti zbrajanja

Primjer. **Asocijativnost** zbrajanja u računalu **ne vrijedi**.

Znamo (odn. uskoro ćete znati) da je tzv. **harmonijski** red

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{i} + \cdots$$

divergentan, tj. suma mu je “**beskonačna**”.

No, nitko nas ne spriječava da računamo konačne početne komade ovog reda, tj. **njegove parcijalne sume**

$$S_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}.$$

A kojim **redom** zbrajamo? (Zbrajanje je **binarna** operacija!)

Primjer neasocijativnosti zbrajanja (nastavak)

U **realnim** brojevima je **potpuno svejedno** kojim poretkom zbrajanja računamo ovu sumu, jer vrijedi **asocijativnost**.

$$a + (b + c) = (a + b) + c = a + b + c.$$

Uostalom, sam zapis izraza **bez zagrada**

$$S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}$$

već “podrazumijeva” **asocijativnost**. U suprotnom, morali bismo **zagradama** naglasiti **poredak** operacija.

Ovdje imamo točno $n - 1$ **binarnih operacija zbrajanja**, i možemo ih napraviti **kojim redom hoćemo**.

Primjer neasocijativnosti zbrajanja (nastavak)

Drugim riječima, u prethodni izraz za S_n

- možemo rasporediti zagrade na bilo koji način, samo da svi plusevi budu “binarni”, tj. zbrajaju dva objekta, a objekt je broj ili (podizraz u zagradama).

Na pr., zbrajanju “unaprijed” odgovara raspored zagrada

$$S_{n,1} := \left(\cdots \left(\left(1 + \frac{1}{2} \right) + \frac{1}{3} \right) + \cdots + \frac{1}{n-1} \right) + \frac{1}{n},$$

a zbrajanju “unatrag” odgovara raspored zagrada

$$S_{n,2} := 1 + \left(\frac{1}{2} + \left(\frac{1}{3} + \cdots + \left(\frac{1}{n-1} + \frac{1}{n} \right) \cdots \right) \right).$$

Primjer neasocijativnosti zbrajanja (nastavak)

Koliko takvih rasporeda zagrada ima — bit će napravljeno u **Diskretnoj matematici**. Bitno je da svi daju **isti rezultat**.

Komutativnost nam uopće **ne treba**. Ako i nju iskoristimo, dobivamo još puno više načina za računanje ove sume, i svi, naravno, opet daju **isti rezultat**.

Izračunajmo **aritmetikom računala** navedene **dvije** sume

• $S_{n,1}$ — unaprijed, i

• $S_{n,2}$ — unatrag,

za $n = 1\,000\,000$, u **tri** standardne IEEE točnosti **single**, **double** i **extended**. Preciznije, koristimo ova tri tipa za prikaz brojeva, uz pripadne aritmetike za računanje.

Primjer neasocijativnosti zbrajanja (nastavak)

Uz skraćene oznake S_1 i S_2 za varijable u kojima zbrajamo pripadne sume, odgovarajući algoritmi za zbrajanje su:

● unaprijed

$$S_1 := 1,$$

$$S_1 := S_1 + \frac{1}{i}, \quad i = 2, \dots, n,$$

● unatrag

$$S_2 := \frac{1}{n},$$

$$S_2 := \frac{1}{i} + S_2, \quad i = n - 1, \dots, 1.$$

Dakle, zaista ne koristimo komutativnost zbrajanja.

Primjer neasocijativnosti zbrajanja (nastavak)

Dobiveni rezultati za sume S_1 , S_2 i pripadne relativne greške su:

tip i suma	vrijednost	rel. greška
single S_1	14.3573579788208008	$2.45740E-0003$
single S_2	14.3926515579223633	$5.22243E-0006$
double S_1	14.3927267228647810	$6.54899E-0014$
double S_2	14.3927267228657545	$-2.14449E-0015$
extended S_1	14.3927267228657234	$1.91639E-0017$
extended S_2	14.3927267228657236	$-1.08475E-0018$

Slovo E u brojevima zadnjeg stupca znači “puta 10 na”, pa je, na pr., $-1.08475E-0018 = -1.08475 \times 10^{-18}$.

Primjer neasocijativnosti zbrajanja (nastavak)

Izračunate vrijednosti S_1 i S_2 su različite (u sve tri točnosti). Dakle, zbrajanje brojeva u aritmetici računala očito **nije asocijativno**.

Primijetite da, u sve tri točnosti, **zbrajanje unatrag** S_2 daje nešto **točniji** rezultat. To **nije slučajno**.

Svi brojevi koje zbrajamo su **istog predznaka** pa zbroj stalno **raste**, bez obzira na poredak zbrajanja.

- Kad zbrajamo unatrag — **od manjih** brojeva **prema većim**, zbroj se **pomalo** “nakuplja”.
- Obratno, kad zbrajamo unaprijed — **od velikih** brojeva **prema manjim**, zbroj puno **brže naraste**. Onda mali dodani član **jedva utječe** na rezultat (tj. dobar dio znamenki pribrojnika nema utjecaj na sumu).

Primjer neasocijativnosti zbrajanja (nastavak)

Drugim riječima, uz isti broj pribrojnika, kod zbrajanja unaprijed

- dodajemo manji broj, ali na veću (dotadašnju) sumu, pa je utjecaj tog pribrojnika bitno manji, a greška veća.

Ovo se izrazito vidi u single S_1 . U usporedbi sa single S_2

- pripadna relativna greška je preko 400 puta veća i dobivamo imamo samo 3 točne znamenke u rezultatu.

Cijeli ovaj eksperiment je napravljen u “prastarom” Turbo Pascalu 7.0 (za DOS) — jer jednostavno podržava sva tri standardna IEEE tipa. Probajte to napraviti u C-u, FORTRAN-u ili nekom drugom programskom jeziku.

Primjer neasocijativnosti zbrajanja (zadatak)

Zadatak. U aritmetici računala, za dovoljno velike n , parcijalne sume $S_{n,1}$ i $S_{n,2}$ postaju **konstantne** (tj. više **ne ovise** o n).

- Zašto? Precizno objasnite!
- Za koji n se to **prvi puta** događa, ovisno o **točnosti** i **smjeru** zbrajanja?
 - Probajte u **single**, pa zaključite što će se dogoditi u **double** i **extended** (bez eksperimentiranja).
- Koji smjer zbrajanja je bolji?

Na kraju ovog primjera, možda je zgodno reći odakle mi “**točan**” rezultat — tj. onaj koji služi za računanje relativnih grešaka u prethodnoj tablici.

Primjer neasocijativnosti zbrajanja (kraj)

Postoji algoritam zbrajanja, tzv. dvostruko kompenzirano zbrajanje (engl. doubly compensated summation), koji uvijek daje zbroj s vrlo malom relativnom greškom (oko $2u$, ako broj pribrojnika nije prevelik). Taj algoritam daje

$$\text{extended } S_{DC} = 14.3927267228657236.$$

Na 18 dekadskih znamenki, rezultat je isti kao i S_2 .

Koga zanima taj i slični algoritmi, može pogledati knjigu:

📖 **Nicholas J. Higham**, *Accuracy and Stability of Numerical Algorithms* (2. ed.), SIAM, Philadelphia, 2002.

Naravno, može se javiti i meni!



Širenje grešaka zaokruživanja (nastavak)

Za analizu grešaka zaokruživanja ne možemo koristiti nikakva “normalna” pravila za aritmetičke operacije u računalu, jer ti zakoni naprosto ne vrijede.

Stvarna algebarska struktura je izrazito komplicirana i postoje debele knjige na tu temu.

- Vrijede neka “zamjenska” pravila, ali su neupotrebljiva za analizu iole većih proračuna.

Međutim, analiza pojedinih operacija postaje bitno lakša, ako uočimo da:

- greške zaokruživanja u aritmetici računala možemo interpretirati i kao egzaktne operacije, ali na “malo” pogrešnim podacima!

Širenje grešaka zaokruživanja (nastavak)

Kako? Dovoljno je faktor $(1 + \varepsilon)$ u ocjeni greške

$$fl(x \circ y) = (1 + \varepsilon)(x \circ y), \quad |\varepsilon| \leq u,$$

“zalijepiti” na x i/ili y . To je isto kao da operand(i) ima(ju) neku relativnu grešku na ulazu u operaciju, a operacija \circ je egzaktna. Dakle,

- izračunati (ili “zaokruženi”) rezultat jednak je egzaktnom, ali za malo promijenjene (tj. perturbirane) podatke (u relativnom smislu).

Što dobivamo ovom interpretacijom?

- Onda možemo koristiti “normalna” pravila aritmetike za analizu grešaka.

Širenje grešaka zaokruživanja (nastavak)

Ne zaboravimo još da ε ovdje ovisi o x , y , i operaciji \circ . Kad takvih operacija ima više, pripadne greške obično označavamo nekim indeksom u ε .

Na primjer, ako je \circ **zbrajanje** (+), onda je

$$\begin{aligned} fl(x + y) &= (1 + \varepsilon_{x+y}) (x + y) \\ &= [(1 + \varepsilon_{x+y}) x] + [(1 + \varepsilon_{x+y}) y], \end{aligned}$$

uz $|\varepsilon_{x+y}| \leq u$, ako su x , y i $x + y$ u prikazivom rasponu.

Potpuno ista stvar vrijedi i za **oduzimanje**.

Kod **množenja** i **dijeljenja** možemo birati kojem ulaznom podatku ćemo “zalijepiti” faktor $(1 + \varepsilon)$.

Širenje grešaka zaokruživanja (nastavak)

Za **množenje** možemo pisati

$$\begin{aligned} fl(x * y) &= (1 + \varepsilon_{x*y}) (x * y) \\ &= [(1 + \varepsilon_{x*y}) x] * y = x * [(1 + \varepsilon_{x*y}) y], \end{aligned}$$

a za **dijeljenje**

$$\begin{aligned} fl(x/y) &= (1 + \varepsilon_{x/y}) (x/y) \\ &= [(1 + \varepsilon_{x/y}) x] / y = x / [y / (1 + \varepsilon_{x/y})]. \end{aligned}$$

Postoje i druge varijante. Na primjer, da svakom operandu “zalijepimo” $\sqrt{1 + \varepsilon}$ (odnosno $1/\sqrt{1 + \varepsilon}$), ali to **nije** naročito važno. **Bitno** je samo da je **izračunati** rezultat **egzaktan** za malo perturbirane podatke.

Širenje grešaka (bilo kojih)

Zasad **nije vidljivo** koja je točno **korist** od ove interpretacije. Stvar se **bolje** vidi tek kad imamo **više operacija zaredom**.

Međutim, ova ideja s “**malo pogrešnim podacima**” je baš ono što nam **treba** za **analizu širenja grešaka** (i to bez obzira na uzrok grešaka), čim se sjetimo da

- rezultati **ranijih** operacija s nekom **greškom ulaze** u nove operacije.

Naime, **uzroka** grešaka može biti mnogo, ovisno o tome što računamo. Od grešaka **modela** i **metode**, preko grešaka **mjerenja** (u ulaznim podacima), do grešaka **zaokruživanja** (ali to je tema za UNM).

Širenje grešaka u aritmetici

Za analizu širenja grešaka u aritmetici, treba pogledati

- što se događa s greškama u rezultatu,
- kad imamo greške u operandima.

Prvo u egzaktnoj aritmetici, a onda i u aritmetici računala.

Pretpostavimo onda da su polazni podaci (ili operandi) x i y malo perturbirani, s pripadnim relativnim greškama ε_x i ε_y .

Koje su operacije opasne (ako takvih ima), ako nam je aritmetika egzaktna, a operandi su $x(1 + \varepsilon_x)$ i $y(1 + \varepsilon_y)$?

Treba ocijeniti relativnu grešku ε_o rezultata operacije \circ

$$(x \circ y)(1 + \varepsilon_o) := [x(1 + \varepsilon_x)] \circ [y(1 + \varepsilon_y)].$$

Širenje grešaka u aritmetici (nastavak)

Naravno, za početak, moramo nešto pretpostaviti o ε_x i ε_y .

Što smatramo **malom** relativnom perturbacijom?

- Svakako **mora** biti $|\varepsilon_x|, |\varepsilon_y| < 1$, inače perturbacijom **gubimo predznak** operanda.

Međutim, to nije dovoljno za neki razuman rezultat.

- Stvarno **očekujemo** $|\varepsilon_x|, |\varepsilon_y| \leq c \ll 1$, tako da imamo barem **nekoliko točnih znamenki** u perturbiranim operandima. Na pr. $c = 10^{-1}$ (jedna točna znamenka).
- **Idealno** u računalu je $|\varepsilon_x|, |\varepsilon_y| \leq u$, tj. kao da smo oba operanda **samo spremili** u memoriju računala (jedna greška zaokruživanja).

Širenje grešaka kod množenja

Množenje je bezopasno (benigno), jer vrijedi

$$\begin{aligned}(x * y) (1 + \varepsilon_*) &:= [x (1 + \varepsilon_x)] * [y (1 + \varepsilon_y)] \\ &= xy (1 + \varepsilon_x + \varepsilon_y + \varepsilon_x \varepsilon_y),\end{aligned}$$

kad stvar napišemo bez nepotrebnih zagrada i *. Onda je

$$\varepsilon_* = \varepsilon_x + \varepsilon_y + \varepsilon_x \varepsilon_y \approx \varepsilon_x + \varepsilon_y,$$

ako su $|\varepsilon_x|$ i $|\varepsilon_y|$ dovoljno mali da $\varepsilon_x \varepsilon_y$ možemo zanemariti.

Dakle, relativna greška se samo zbraja.

U idealnom slučaju $|\varepsilon_x|, |\varepsilon_y| \leq u$, dobivamo približnu ocjenu relativne greške $|\varepsilon_*| \leq 2u$ (do na u^2), ili, na pr., $|\varepsilon_*| \leq 2.01u$.

Širenje grešaka kod dijeljenja

Dijeljenje je, također, bezopasno (benigno), samo je zaključak malo dulji. Na početku je

$$(x/y) (1 + \varepsilon_x) := [x (1 + \varepsilon_x)] / [y (1 + \varepsilon_y)] = \frac{x (1 + \varepsilon_x)}{y (1 + \varepsilon_y)}.$$

Ako su $|\varepsilon_x|$ i $|\varepsilon_y|$ dovoljno mali da sve možemo linearizirati (tj. zanemariti “kvadratne” i više potencije epsilon), onda je

$$\frac{1}{1 + \varepsilon_y} = 1 - \varepsilon_y + \sum_{n=2}^{\infty} (-1)^n \varepsilon_y^n \approx 1 - \varepsilon_y$$

i

$$(1 + \varepsilon_x) (1 - \varepsilon_y) = 1 + \varepsilon_x - \varepsilon_y - \varepsilon_x \varepsilon_y \approx 1 + \varepsilon_x - \varepsilon_y.$$

Širenje grešaka kod dijeljenja (nastavak)

Kad to uvrstimo u prvi izraz, dobivamo

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

Za relativnu grešku (približno) vrijedi

$$\varepsilon_/ \approx \varepsilon_x - \varepsilon_y, \quad |\varepsilon_/| \approx |\varepsilon_x| + |\varepsilon_y|.$$

Dakle, relativne greške se oduzimaju, a ocjene zbrajaju.

U idealnom slučaju $|\varepsilon_x|, |\varepsilon_y| \leq u$, opet dobivamo približnu ocjenu relativne greške $|\varepsilon_/| \leq 2u$.

Vidimo da su i množenje i dijeljenje bezopasne operacije za širenje grešaka zaokruživanja.

Širenje grešaka kod zbrajanja i oduzimanja

Zbrajanje i oduzimanje. Ovdje rezultat ključno ovisi o predznacima od x i y .

Sasvim općenito, neka su x i y proizvoljnih predznaka. Za zbrajanje i oduzimanje (oznaka \pm) vrijedi

$$\begin{aligned}(x \pm y) (1 + \varepsilon_{\pm}) &:= [x (1 + \varepsilon_x)] \pm [y (1 + \varepsilon_y)] \\ &= x (1 + \varepsilon_x) \pm y (1 + \varepsilon_y) = (x \pm y) + (x \varepsilon_x \pm y \varepsilon_y).\end{aligned}$$

Pogledajmo prvo trivijalne slučajeve. Ako je egzaktan rezultat $x \pm y = 0$, onda imamo dvije mogućnosti.

- Ako je i $x (1 + \varepsilon_x) \pm y (1 + \varepsilon_y) = 0$, relativna greška ε_{\pm} može biti koji broj (nije određena), a prirodno je uzeti $\varepsilon_{\pm} = 0$.

Širenje grešaka kod zbrajanja i oduzimanja

- U protivnom, za $x(1 + \varepsilon_x) \pm y(1 + \varepsilon_y) \neq 0$, gornja jednakost je nemoguća, pa stavljamo $\varepsilon_{\pm} = \pm\infty$.

Pretpostavimo nadalje da je $x \pm y \neq 0$. Onda je

$$\begin{aligned}(x \pm y)(1 + \varepsilon_{\pm}) &= (x \pm y) + (x\varepsilon_x \pm y\varepsilon_y) \\ &= (x \pm y) \left(1 + \frac{x\varepsilon_x \pm y\varepsilon_y}{x \pm y} \right).\end{aligned}$$

Relativnu grešku ε_{\pm} možemo napisati u obliku linearne kombinacije polaznih grešaka ε_x i ε_y

$$\varepsilon_{\pm} = \frac{x\varepsilon_x \pm y\varepsilon_y}{x \pm y} = \frac{x}{x \pm y} \varepsilon_x \pm \frac{y}{x \pm y} \varepsilon_y.$$

Širenje grešaka kod zbrajanja i oduzimanja

Naravno, za nastavak rasprave ključno je pitanje

• koliko su **veliki faktori** uz polazne greške (da li “**prigušuju**” ili “**napuhavaju**” greške).

Da ne bismo stalno pisali hrpu oznaka \pm (nepregledno), pogledajmo što se zbiva kad

• x i y imaju **isti** predznak, a

• **posebno** gledamo operacije $+$ i $-$.

Ako su x i y različitih predznaka, zamijenimo operaciju u suprotnu ($+$ \mapsto $-$, $-$ \mapsto $+$), pa će vrijediti isti zaključci.

Dakle, zbrajamo i oduzimamo brojeve **istih** predznaka.

Širenje grešaka kod zbrajanja

Zbrajanje brojeva istog predznaka je bezopasno (benigno). To izlazi ovako.

Zbog istih predznaka vrijedi $|x|, |y| \leq |x + y|$, pa je

$$\left| \frac{x}{x + y} \right|, \left| \frac{y}{x + y} \right| \leq 1.$$

Odavde odmah slijedi

$$|\varepsilon_+| \leq |\varepsilon_x| + |\varepsilon_y|.$$

Dakle, relativna greška se, u najgorem slučaju, zbraja.

U idealnom slučaju $|\varepsilon_x|, |\varepsilon_y| \leq u$, opet dobivamo ocjenu relativne greške $|\varepsilon_+| \leq 2u$.

Širenje grešaka kod zbrajanja (nastavak)

Uz malo truda, dobivamo i **bolju** ocjenu. Prvo uočimo da za faktore vrijedi

$$\left| \frac{x}{x+y} \right| + \left| \frac{y}{x+y} \right| = 1,$$

i još iskoristimo $|\varepsilon_x|, |\varepsilon_y| \leq \max\{|\varepsilon_x|, |\varepsilon_y|\}$. Onda je

$$\begin{aligned} |\varepsilon_+| &\leq \left| \frac{x}{x+y} \right| |\varepsilon_x| + \left| \frac{y}{x+y} \right| |\varepsilon_y| \\ &\leq \left(\left| \frac{x}{x+y} \right| + \left| \frac{y}{x+y} \right| \right) \max\{|\varepsilon_x|, |\varepsilon_y|\} \\ &= \max\{|\varepsilon_x|, |\varepsilon_y|\}. \end{aligned}$$

Širenje grešaka kod zbrajanja (nastavak)

Dakle, relativna greška zbrajanja je, u najgorem slučaju, **maksimum** polaznih grešaka (ne treba zbrajati).

U idealnom slučaju $|\varepsilon_x|, |\varepsilon_y| \leq u$, sada dobivamo ocjenu relativne greške $|\varepsilon_+| \leq u$. Bolje ne može!

Naravno, isto vrijedi i za **oduzimanje** brojeva **različitih** predznaka. I to je **bezopasno**.

Širenje grešaka kod oduzimanja

Oduzimanje brojeva istog predznaka može biti opasno (čak katastrofalno loše).

☞ Točnije, ne mora uvijek biti opasno, ali može!

Zašto i kada je opasno?

Zbog različitih predznaka od x i y sigurno vrijedi

$$|x - y| \leq \max\{|x|, |y|\},$$

pa je barem jedan od faktora veći od 1, tj.

$$\max \left\{ \left| \frac{x}{x - y} \right|, \left| \frac{y}{x - y} \right| \right\} > 1.$$

Širenje grešaka kod oduzimanja (nastavak)

Odavde odmah slijedi da u ocjeni relativne greške

$$|\varepsilon_-| \leq \left| \frac{x}{x-y} \right| |\varepsilon_x| + \left| \frac{y}{x-y} \right| |\varepsilon_y|$$

na barem **jednom** mjestu imamo **rast** greške, a može i na **oba** mjesta.

Kad je to **zaista opasno**? Ako je $|x-y| \ll |x|, |y|$, ovi faktori

$$\left| \frac{x}{x-y} \right|, \left| \frac{y}{x-y} \right|,$$

mogu biti **proizvoljno veliki**, pa i relativna greška $|\varepsilon_-|$ rezultata može biti **proizvoljno velika**!

Opasno oduzimanje ili kraćenje

Opasna situacija $|x - y| \ll |x|, |y|$ znači da je

- rezultat oduzimanja brojeva istog predznaka =
- broj koji je po apsolutnoj vrijednosti **mного manji** od polaznih podataka (oba operanda),

a to znači da operandi x i y **moraju** biti **bliski**, tako da dolazi do **kraćenja**. Zato se ovaj **fenomen** obično zove

Opasno ili katasrofalno kraćenje.

Dosad smo govorili da relativna greška u tom slučaju **može** biti **velika**, ali da li se to **zaista događa**?

Naime, ovdje je ipak riječ o **ocjeni** greške, pa se možda događa da je **ocjena vrlo loša**, a prava **greška** ipak **mala**!

Primjer katastrofalnog kraćenja

Nažalost, **nije tako!** To se itekako **događa u praksi!**

Primjer. Uzmimo realnu aritmetiku “računala” u bazi 10. Za mantisu (značajni dio) imamo $t = 4$ dekadске znamenke, a za eksponent $s = 2$ znamenke (što nije bitno). Neka je

$$x = 8.8866 = 8.8866 \times 10^0,$$

$$y = 8.8844 = 8.8844 \times 10^0.$$

Umjesto brojeva x i y (koji nisu prikazivi), u “memoriju” spremamo brojeve $fl(x)$ i $fl(y)$, pravilno zaokružene na $t = 4$ znamenke

$$fl(x) = 8.887 \times 10^0,$$

$$fl(y) = 8.884 \times 10^0.$$

Primjer katastrofalnog kraćenja (nastavak)

Ovim zaokruživanjem smo napravili **malu** relativnu grešku (ovdje je $u = 5 \times 10^{-5}$).

Razliku $f_l(x) - f_l(y)$ računamo tako da **izjednačimo eksponente** (što već jesu), **oduzmemo** značajne dijelove (mantise), pa **normaliziramo**

$$\begin{aligned} f_l(x) - f_l(y) &= 8.887 \times 10^0 - 8.884 \times 10^0 \\ &= 0.003 \times 10^0 = 3.??? \times 10^{-3}. \end{aligned}$$

Kod normalizacije, zbog pomaka “**ulijevo**”, pojavljuju se

● **?** = znamenke koje više **ne možemo** restaurirati (ta informacija se izgubila).

Što sad?

Primjer katastrofalnog kraćenja (nastavak)

Računalo radi **isto** što bismo i mi napravili:

👉 na ta mjesta ? upisuje 0.

Razlog: da rezultat bude **točan**, ako su ulazni brojevi točni. Dakle, ovo oduzimanje je **egzaktno** i u aritmetici računala.

Konačni rezultat je $fl(x) - fl(y) = 3.000 \times 10^{-3}$.

Pravi rezultat je

$$\begin{aligned}x - y &= 8.8866 \times 10^0 - 8.8844 \times 10^0 \\ &= 0.0022 \times 10^0 = 2.2 \times 10^{-3}.\end{aligned}$$

Već **prva** značajna znamenka u $fl(x) - fl(y)$ je **pogrešna**, a relativna greška **ogromna**! Uočite da je ta znamenka (**3**) ujedno i **jedina** koja nam je ostala — sve ostalo se skratilo!

Primjer katastrofalnog kraćenja (nastavak)

Prava **katastrofa** se događa ako $3.??? \times 10^{-3}$ uđe u naredna zbrajanja (oduzimanja), a onda se **skrati** i ta trojka!

Uočite da je **oduzimanje** $fl(x) - fl(y)$ bilo **egzaktno** (a egzaktno je i u aritmetici računala), ali **rezultat je pogrešan**.

Krivac, očito, nije **oduzimanje** (kad je egzaktno).

- Uzrok su **polazne greške** u operandima.

Ako njih **nema**, tj. ako su operandi **egzaktni**,

- i dalje (naravno) dolazi do **kraćenja**,

- ali je rezultat (uglavnom, a po IEEE standardu sigurno) **egzaktan**,

pa se ovo kraćenje zove **benigno kraćenje**.

Širenje grešaka u aritmetici računala

Dosad smo gledali širenja grešaka u egzaktnoj aritmetici.

U aritmetici računala postupamo na potpuno isti način. Samo treba zgodno iskoristiti onu raniju interpretaciju da je

- izračunati (ili “zaokruženi”) rezultat jednak egzaktnom, ali za malo perturbirane podatke (u relativnom smislu).

A širenje grešaka u egzaktnoj aritmetici znamo.

Svaka aritmetička operacija u računalu samo povećava perturbaciju ulaznih podataka za jedan faktor oblika $(1 + \varepsilon)$, uz ocjenu $|\varepsilon| \leq u$, ovisno o tome kojim operandima “zalijepimo” taj faktor.

Širenje grešaka u aritmetici računala

Na pr., uzmimo da računamo zbroj $x + y$, gdje su x i y spremljeni u računalu. Znamo da za izračunati rezultat vrijedi

$$\begin{aligned} fl(x + y) &= (1 + \varepsilon_{x+y}) (x + y) \\ &= [(1 + \varepsilon_{x+y}) x] + [(1 + \varepsilon_{x+y}) y], \end{aligned}$$

uz $|\varepsilon_{x+y}| \leq u$, ako su x , y i $x + y$ u prikazivom rasponu.

No, x i y već imaju neke greške obzirom na prave egzaktne vrijednosti. I to treba uvrstiti u ovu formulu.

Natuknice o analizi grešaka

- Bilo koja pojedina **operacija**, **nakon**, recimo prvog čitanja, ili ranijih operacija — tad ide kao ovo gore, ali svagdje imam **dodatni faktor** oblika $(1 + \varepsilon)$ iz ocjene greške (ako je sve prikazivo u dozvoljenom rasponu).
- Dakle, faktori se “kote” i postoje razne oznake za to, da se lakše čita.
 - oznake: ε (s indeksima) — obično za “jedinične” greške (ispod u),
 - neko drugo slovo (na pr. η) s indeksima, za ostale (relativne) greške u analizi.

Natuknice o analizi grešaka (nastavak)

Pojmovi greške **unaprijed** (forward error), i greške **unatrag** (katkad se zove i **obratna** — backward error).

- Gledam algoritam kao preslikavanje: ulaz (domena) u izlaz (kodomena).
- Zanima me greška u rezultatu (kodomenu) — forward.
- To katkad ide, ali je, uglavnom, teško (ili daje loše ocjene). (Primjer - Zlatko, za normu u \mathbb{R}^2 , i još dodaj scaling).
- Lakše je “unatrag” — ista interpretacija kao i za pojedine operacije.

Uočiti da se **akumulacija** faktora $(1 + \varepsilon)$ **prirodno** radi **unatrag** — inače moram znati grešku za operande (a to je rezultat **unaprijed**).

Natuknice o analizi grešaka (nastavak)

Postupak “unatrag”:

- Krajnji rezultati algoritma su “egzaktni” ali na ponešto perturbiranim podacima (i napravi se ocjena tih perturbacija u **domeni**).
- A zatim ide matematička **teorija perturbacije**, koja daje ocjene u **kodomeni** (izvod ide za egzaktni račun, pa vrijede normalna pravila).

I sad imam pojmove: **stabilno** i **nestabilno** računanje (algoritam) (“prigušivač” ili “pojačalo” grešaka).

- Slikice (skripta NA, Higham).
- Primjeri nestabilnosti — **uklonjivi** i **NEuklonjivi**.

Dodatna literatura za floating point aritmetiku

Ako želite saznati još ponešto o floating–point prikazu brojeva i aritmetici, pogledajte:

- **David Goldberg**, *What Every Computer Scientist Should Know About Floating–Point Arithmetic*, ACM Computing Surveys, Vol. 23, No. 1, March 1991, pp. 5–48.
- **Zlatko Drmač**, *Numerička matematika i računala*, MFL 4/196, Zagreb, 1999., str. 212–219.

i već spomenutu **Highamovu** knjigu.

Zbog “copyrighta”, ovo nije na mom webu, ali možete dobiti, ako želite.

Goldbergov članak je na webu (samo ja za to “ne znam”): pod “linkovi”, zadnji link.

Kvadratna jednadžba

Uzmimo da treba riješiti (realnu) kvadratnu jednadžbu

$$ax^2 + bx + c = 0,$$

gdje su a , b i c zadani i $a \neq 0$.

Matematički gledano, problem je lagan: imamo 2 rješenja

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Numerički gledano, problem je mnogo izazovniji:

- ni uspješno računanje po ovoj formuli,
- ni točnost izračunatih korijena,

ne možemo uzeti “zdravo za gotovo”.

Kvadratna jednačba (nastavak)

Primjer: $x^2 - 56x + 1 = 0$. U aritmetici s 5 decimala dobijemo

$$x_1 = \frac{56 - \sqrt{3132}}{2} = \frac{56 - 55.964}{2} = 0.018000,$$

$$x_2 = \frac{56 + \sqrt{3132}}{2} = \frac{56 + 55.964}{2} = 55.982.$$

Točna rješenja su

$$x_1 = 0.0178628\dots \quad \text{i} \quad x_2 = 55.982137\dots$$

Manji od ova dva korijena ima samo dvije točne znamenke (kraćenje).

Kvadratna jednadžba (popravak)

Prvo izračunamo većeg po apsolutnoj vrijednosti, po formuli

$$x_2 = \frac{-(b + \text{sign}(b)\sqrt{b^2 - 4ac})}{2a},$$

a manjeg po apsolutnoj vrijednosti, izračunamo iz

$$x_1 \cdot x_2 = \frac{c}{a}$$

(Vieta), tj.

$$x_1 = \frac{c}{x_2 a}.$$

Opasnog kraćenja za x_1 više nema!

Kvadratna jednadžba (nastavak)

Ovo je bila samo **jedna**, od (barem) **tri** “opasne” točke za računanje. Preostale dvije su:

- 🔴 “kvadriranje” pod korijenom — mogućnost za **overflow**.
Rješenje — “skaliranjem”.
- 🔴 oduzimanje u diskriminanti (kraćenje) — nema jednostavnog rješenja.
- 🔴 To je odraz **nestabilnosti** problema, jer tad imamo **dva bliska korijena** koji su **osjetljivi** na male perturbacije koeficijenata (na pr. pomak c “gore–dolje”).

Neki primjeri izbjegavanja kraćenja

Primjer. Treba izračunati

$$y = \sqrt{x + \delta} - \sqrt{x},$$

gdje su x i δ zadani ulazni podaci, s tim da je $x > 0$,

• a $|\delta|$ vrlo mali broj.

U ovoj formuli, očito, dolazi do velike greške zbog kraćenja — zaokruživanje korijena prije oduzimanja.

Ako formulu “deracionaliziramo” u oblik

$$y = \frac{\delta}{\sqrt{x + \delta} + \sqrt{x}},$$

problema više nema!

Neki primjeri izbjegavanja kraćenja

Primjer. Treba izračunati

$$y = \cos(x + \delta) - \cos x,$$

gdje su x i δ zadani **ulazni** podaci, s tim da je $|\cos x|$ razumno velik,

• a $|\delta|$ **vrlo mali** broj.

Opet, dolazi do **velike greške** zbog **kraćenja**.

Ako formulu napišmo u “**produktnom**” obliku

$$y = -2 \sin \frac{\delta}{2} \sin \left(x + \frac{\delta}{2} \right),$$

problema više **nema!**