Numerička matematika
1. predavanje — dodatak

Saša Singer

Sadržaj predavanja — dodatka

- Prikaz realnih brojeva u računalu — IEEE standard:
  - Osnovni oblik "floating–point" prikaza — mantisa i eksponent.
  - IEEE standard — tipovi: single, double, extended.
  - Greške zaokruživanja u prikazu.
  - Pojam "jedinične greške zaokruživanja".
- 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:

$$67800000000.0 \qquad 0.000002078$$

Koristimo tzv. "znanstveni" zapis (ili notaciju), u kojem

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

Dogovor: vodeći dio je jednoznamenkast ispred točke!

U bazi 10, to znači između 1 i 10 (strogo ispod 10). Nakon odgovarajućih pomaka decimalne točke, znanstveni zapisi su:

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

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. tako da vrijedi

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

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

- prikaziv je samo neki raspon realnih brojeva,
- neki brojevi unutar prikazivog raspona nisu prikazivi, jer im je mantisa predugačka $\Longrightarrow$ zaokruživanje.

Prikaz realnih brojeva (nastavak)

Primjer. Znanstveni zapis brojeva u binarnom sustavu:

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

$$0.0001011011 = 1.011011 \cdot 2^{-4}$$

Primijetite da se vodeća jedinica u normaliziranom obliku ne mora pamtiti, ako znamo da je broj $\neq 0$. U tom slučaju,

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

Tada se vodeća jedinica zove skriveni bit (engl. hidden bit), jer se ne pamti, nego se "podrazumijeva".

Ipak, ovo je samo pojednostavljeni prikaz realnih brojeva. Stvarni prikaz je malo složeniji.

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, označimo ga s e,

- dodaje konstanta — takva da je "pomaknuti" eksponent uvijek pozitivan, za normalizirane brojeve.

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

- recipročna vrijednost najmanjeg pozitivnog normaliziranog broja.

Takav "pomaknuti" eksponent naziva se karakteristika, a normalizirana mantisa obično se zove signifikand.

Stvarni prikaz realnih brojeva — IEEE 754

Stvarni prikaz realnih brojeva ima tri dijela i svaki od njih ima svoju duljinu — broj bitova predviđenih za prikaz tog dijela.

- predznak s — uvijek zauzima jedan bit, i to najviši;
- karakteristika k — zauzima sljedećih w bitova (w = engl. "width", širina pomaknutog eksponenta);
- signifikand m — zauzima sljedećih t bitova (t = engl. "trailing", završni ili razlomljeni dio od m).

Po starom standardu — ako se pamti vodeći (cjelobrojni) bit mantise, on je prvi (vodeći) u m, a duljina je $t + 1$.

Još se koristi i standardna oznaka

- preciznost $p := t + 1$ — to je ukupni broj vodećih značajnih bitova cijele mantise.

Stvarni prikaz realnih brojeva — IEEE 754

Karakteristika k se interpretira kao cijeli broj bez predznaka, tako da je $k \in \{0, \ldots, 2^w - 1\}$. "Rubne" vrijednosti za k označavaju tzv. posebna stanja:

- $k = 0$ — nula i denormalizirani brojevi,
- $k = 2^w - 1$ — beskonačno (`Inf`) i "nije broj" (`NaN`).

Sve ostale vrijednosti $k \in \{1, \ldots, 2^w - 2\}$ koriste se za prikaz normaliziranih brojeva različitih od nule.

Veza između karakteristike k i stvarnog eksponenta e je:

$$k = e + bias, \qquad bias = 2^{w-1} - 1.$$

Dakle, dozvoljeni eksponenti e moraju biti između

$$e_{\min} = -(2^{w-1} - 2) \quad \text{i} \quad e_{\max} = 2^{w-1} - 1.$$

Standardni tipovi realnih brojeva — IEEE 754

Novi standard IEEE 754-2008 standard ima sljedeće tipove za prikaz realnih brojeva:

`ime tipa`	`binary32`	`binary64`	`binary128`
duljina u bitovima	32	64	128
$t =$	23	52	112
$w =$	8	11	15
$u = 2^{-p}$	2^{-24}	2^{-53}	2^{-113}
$u \approx$	$5.96 \cdot 10^{-8}$	$1.11 \cdot 10^{-16}$	$9.63 \cdot 10^{-35}$
raspon brojeva $\approx$	$10^{\pm 38}$	$10^{\pm 308}$	$10^{\pm 4932}$

Broj u je tzv. jedinična greška zaokruživanja (v. malo kasnije).

Najveći tip `binary128` još uvijek ne postoji u većini procesora.

Standardni tipovi realnih brojeva — extended

Većina PC procesora još uvijek ima posebni dio — tzv. FPU (engl. Floating–Point Unit). On stvarno koristi

- tip `extended` iz starog standarda, koji odgovara tipu `extended binary64` u novom IEEE 754-2008 standardu.

Dio primjera koje ćete vidjeti napravljen je baš u tom tipu!

`ime tipa`	`extended`
duljina u bitovima	80
$t+1=$	$63+1$
$w=$	15
$u=2^{-p}$	2^{-64}
$u\approx$	$5.42\cdot10^{-20}$
raspon brojeva $\approx$	$10^{\pm4932}$

Oznake

Oznake:

- Crveno — duljina odgovarajućeg polja u bitovima, bitove brojimo od 0, zdesna nalijevo (kao i obično),
- s — predznak: 0 za pozitivan broj, 1 za negativan broj,
- k — karakteristika,
- m — mantisa (signifikand).

1	w	t
s	k	m

- Najznačajniji bit u odgovarajućem polju je najljeviji, a najmanje značajan bit je najdesniji.

Stvarni prikaz tipa single (binary32)

“Najkraći” realni tip je tzv. realni broj jednostruke točnosti. U C-u se taj tip zove `float`. Savjet: ne koristiti u praksi!

On ima sljedeća svojstva:

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

1	8	23
s	k	m

- u mantisi se ne pamti vodeća jedinica, ako je broj normaliziran,
- stvarni eksponent e broja, $e \in \{-126, \ldots, 127\}$,
- karakteristika $k = e + 127$, tako da je $k \in \{1, \ldots, 254\}$,
- 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.

$$(10.25)_{10} = \left(10 + \frac{1}{4}\right)_{10} = (10 + 2^{-2})_{10}$$

$$= (1010.01)_2 = 1.01001 \cdot 2^3.$$

Prema tome je:

$$s = 0$$

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

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

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

Realni broj nula ima dva prikaza:

- mantisa i karakteristika imaju sve bitove jednake 0,

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 bit mantise koji nije nula, onda se kao eksponent e uzima -126 = najmanji dozvoljeni.

Mantisa takvog broja nije normalizirana i počinje s $0.m$.

Takvi brojevi zovu se denormalizirani brojevi.

Primjer. Kako izgleda prikaz binarno zapisanog realnog broja

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

Rješenje:

$$s = 0$$

$$k = 0000\,0000$$

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

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

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

- $s = 0$ — prikaz $+\infty$, skraćena oznaka `+Inf`,
- $s = 1$ — prikaz $-\infty$, skraćena oznaka `-Inf`.

Rezultat `Inf` (odnosno, `-Inf`) dobivamo ako

- pokušamo spremiti preveliki broj (tzv. "overflow"), ili
- ako nešto različito od nule podijelimo s nulom.

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

$$s = 0 \qquad (s = 1)$$

$$k = 1111\ 1111$$

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

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

Ako je $k = 255$ i postoji bar jedan bit mantise različit od nule, onda je to oznaka za

- tzv. "Not a Number" ("nije broj") ili, skraćeno, `NaN`.

Rezultat `NaN` je uvijek signal da se radi o pogrešci. Na pr.,

- dijeljenje nule s nulom,
- vađenje drugog korijena iz negativnog broja i sl.

Primjer.

$$
\begin{aligned}
s &= 0 \\
k &= 1111\ 1111 \\
m &= 000\ 0000\ 0000\ 0101\ 0000\ 0000
\end{aligned}
$$

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)

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

ima 25 znamenki mantise i ne može se egzaktno spremiti u realni broj jednostruke točnosti, odnosno, tip `float` u C-u, koji ima $23 + 1$ znamenki za mantisu. Što se onda zbiva?

Tada se pronalaze dva najbliža prikaziva susjeda B_-, B_+, broju B, takva da vrijedi

$$B_- < B < B_+.$$

Greške zaokruživanja (nastavak)

U našem primjeru je:

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

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

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

Nakon toga, zaokružuje se rezultat. Zaokruživanje može biti:

- prema najbližem broju (standardno, engl. "default", za sve procesore) — ako su dva susjeda jednako udaljena od B, izabire "parni" od ta dva broja $\iff$ zadnji bit je 0,
- prema dolje, tj. prema $-\infty$,
- prema gore, tj. prema $+\infty$,
- prema nuli, tj. odbacivanjem "viška" znamenki.

Greške zaokruživanja (nastavak)

Standardno zaokruživanje u našem primjeru:

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

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

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

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

Način zaokruživanja (tzv. "rounding mode") može se birati

- postavljanjem tzv. "procesorskih zastavica", ili opcijama za compiler.

U nastavku pretpostavljamo standardno zaokruživanje.

- Za ostala tri načina, ocjena za grešku je 2 puta veća!

Jedinična greška zaokruživanja — standardno

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

Time smo napravili grešku zaokruživanja $\leq \frac{1}{2}$ "zadnjeg bita" mantise (tj. $\leq \frac{1}{2} 2^{-t} = 2^{-t-1} = 2^{-p}$). Ta gornja ocjena 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

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

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:

1	8	23
s	k	m

Vrijednost broja je

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

Raspon tipa float

Najveći prikazivi pozitivni broj je

$$\texttt{FLT\_MAX} = (1 - 2^{-24}) \cdot 2^{128} \approx 3.40282347 \cdot 10^{38},$$

s prikazom

$$\begin{aligned} s &= 0 \\ k &= 1111\ 1110 \\ m &= 111\ 1111\ 1111\ 1111\ 1111\ 1111 \end{aligned}$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\texttt{FLT\_MIN} = 2^{-126} \approx 1.17549435 \cdot 10^{-38},$$

s prikazom

$$\begin{aligned} s &= 0 \\ k &= 0000\ 0001 \\ m &= 000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{aligned}$$

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/FLT_MIN je egzaktno prikaziv (nađite prikaz),
- 1/FLT_MAX nije egzaktno prikaziv i zalazi u denormalizirane brojeve (tzv. "gradual underflow").

Najmanji prikazivi denormalizirani pozitivni broj je $2^{-126} \cdot 2^{-23} = 2^{-149} \approx 1.40129846 \cdot 10^{-45}$, s prikazom

$$\begin{aligned} s &= 0 \\ k &= 0000\ 0000 \\ m &= 000\ 0000\ 0000\ 0000\ 0000\ 0001 \end{aligned}$$

Stvarni prikaz tipa double (binary64)

"Srednji" realni tip je tzv. realni broj dvostruke točnosti. U C-u se taj tip zove `double`. Savjet: njega treba koristiti!

On ima sljedeća svojstva:

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

1	11	52
s	k	m

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

Prikaz brojeva dvostruke točnosti — sažetak

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

1	11	52
s	k	m

Vrijednost broja je

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

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

$$\texttt{DBL\_EPSILON} = 2u \approx 2.2204460492503131 \cdot 10^{-16}.$$

Najveći prikazivi pozitivni broj je

$$\texttt{DBL\_MAX} = (1 - 2^{-53}) \cdot 2^{1024} \approx 1.7976931348623157 \cdot 10^{308}.$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\texttt{DBL\_MIN} = 2^{-1022} \approx 2.2250738585072014 \cdot 10^{-308}.$$

Tip extended

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

On ima sljedeća svojstva:

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

1	15	1	63
s	k	i	m

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

Prikaz brojeva proširene točnosti — sažetak

IEEE tip extended:

1	15	1	63
s	k	i	m

s tim da je $i = 0 \Longleftrightarrow k = 0$ (tu je redundantnost u prikazu).

Vrijednost broja je

$$v = \begin{cases} (-1)^s * 2^{(k-16383)} * (1.m) & \text{ako je } 0 < k < 32767, \\ (-1)^s * 2^{(-16382)} * (0.m) & \text{ako je } k = 0 \text{ i } m \neq 0, \\ (-1)^s * 0 & \text{ako je } k = 0 \text{ i } m = 0, \\ (-1)^s * \texttt{Inf} & \text{ako je } k = 32767 \text{ i } m = 0, \\ \texttt{NaN} & \text{ako je } k = 32767 \text{ i } m \neq 0. \end{cases}$$

Realna aritmetika računala (IEEE standard)

Zbrajanje realnih brojeva u računalu

Primjer. Uzmimo "računalo" u bazi 10, s $p = 4$ značajne dekadske znamenke. Treba naći rezultat zbrajanja brojeva

$$x = 9.937 \times 10^0, \quad y = 8.165 \times 10^{-2}.$$

Prvo se izjednače eksponenti na onaj veći, uz pomak mantise manjeg broja udesno (ako treba), a onda se zbroje te mantise. Dobiveni rezultat se normalizira (ako treba) i zaokružuje.

$$\begin{array}{rll}
x = & 9.937 & \times 10^0 \\
y = & 0.08165 & \times 10^0 \quad \leftarrow \text{pomak} \\
\hline
x + y = & 10.01865 & \times 10^0 \quad \leftarrow \text{zbroj} \\
 & & \\
x + y = & 1.001865 & \times 10^1 \quad \leftarrow \text{normalizacija} \\
f\ell(x + y) = & 1.002 & \times 10^1 \quad \leftarrow \text{zaokruživanje}
\end{array}$$

■

Oznake

Oznaka. Neka je $\mathcal{F}$ skup svih realnih brojeva koji se nalaze u dozvoljenom rasponu za normalizirani prikaz (u danom tipu).

Što to znači? Broj $x \in \mathcal{F}$ ne mora biti prikaziv, ali njegova

- zaokružena aproksimacija $f\ell(x)$ ima normalizirani prikaz,

pa onda i malu relativnu grešku.

Najmanji normalizirani prikazivi broj u danom tipu je

$$v_{\min} = [\,1.000\ldots 0\,] \cdot 2^{e_{\min}} = 2^{e_{\min}},$$

a najveći normalizirani prikazivi broj je

$$\begin{aligned} v_{\max} &= [\,1.111\ldots 1\,] \cdot 2^{e_{\max}} = (1 + (1 - 2^{-t})) \cdot 2^{e_{\max}} \\ &= 2(1 - 2^{-t-1}) \cdot 2^{e_{\max}} = (1 - 2^{-p}) \cdot 2^{e_{\max}+1}. \end{aligned}$$

Dakle, vrijedi $x \in \mathcal{F} \iff |f\ell(x)| \in [v_{\min}, v_{\max}]$.

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 $\sqrt{\ }$, ali ne mora vrijediti za sve funkcije (na pr. za $\sin$ oko 0, ili $\ln$ oko 1).

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

$$f\ell(x \circ y) = (1 + \varepsilon)\,(x \circ y), \qquad |\varepsilon| \leq u.$$

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 (egzaktni rezultati imaju konačan binarni prikaz),
- ali kod dijeljenja to sigurno ne ide (egzaktni 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 "mjesta za razlike" nema puno, tj. razlike su zaista rijetke.

Napomena. Oznaka $f\ell(izraz)$, 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 egzaktni 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,
- egzaktni 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 $\sqrt{\ }$),
- ili dijelimo s nulom, pa dobivamo `Inf` ili `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 egzaktni rezultat ionako ne znamo.

Općenito, može nam se dogoditi da je egzaktni 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 (egzaktni) rezultat izvan prikazivog raspona (na pr. x^2).

Iako egzaktni 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

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 ili širenja 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 $\mathbb{R}$.

Na primjer, za aritmetičke operacije u računalu

- nema asocijativnosti zbrajanja i množenja,
- nema distributivnosti množenja prema zbrajanju.

Dakle, poredak izvršavanja operacija je bitan!

Zapravo, jedino standardno pravilo koje vrijedi je

- komutativnost za zbrajanje i za množenje.

(Relativno nedavno je postojalo računalo na kojem ni to nije vrijedilo — prva Cray računala).

Primjer:
Neasocijativnost zbrajanja

Primjer neasocijativnosti zbrajanja

Primjer. Asocijativnost zbrajanja u računalu ne vrijedi.

Znamo (odnosno, 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 (tzv. Catalanovi brojevi). Bitno nam je samo da svi ti rasporedi, matematički, 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:

$$
\begin{aligned}
S_1 &:= 1, \\
S_1 &:= S_1 + \frac{1}{i}, \quad i = 2, \ldots, n,
\end{aligned}
$$

- unatrag:

$$
\begin{aligned}
S_2 &:= \frac{1}{n}, \\
S_2 &:= \frac{1}{i} + S_2, \quad i = n-1, \ldots, 1.
\end{aligned}
$$

Dakle, zaista ne koristimo komutativnost zbrajanja.

Primjer neasocijativnosti zbrajanja (nastavak)

Dobiveni rezultati za sume S_1, S_2 (prva pogrešna znamenka je crvena) i pripadne relativne greške su:

`tip i suma`	`vrijednost`	`rel. greška`
single S_1	14.3**5**7357978820800781 2	2.45740E−03
single S_2	14.392**6**5155792236328 12	5.22243E−06
double S_1	14.39272672286**4**7810526	6.54899E−14
double S_2	14.392726722865**7**544962	−2.14449E−15
extended S_1	14.39272672286572**3**3553	1.91639E−17
extended S_2	14.392726722865723**6**467	−1.08475E−18

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

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. Pred kraj, 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 samo 3 točne znamenke u rezultatu.

Cijeli ovaj eksperiment napravljen je u "prastarom" Turbo Pascalu 7.0 (DOS), danas Delphi — 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

$$\text{extended}\ \ S_{DC} = 14.392726722865723631 1.$$

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

Koga zanima taj i slični algoritmi, može pogledati knjigu:

- Nicholas J. Higham, Accuracy and Stability of Numerical Algorthms (2. ed.), SIAM, Philadelphia, 2002.

Naravno, može se javiti i meni! ■

Širenje grešaka u aritmetici

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 rezultatu, ali za malo promijenjene (tj. perturbirane) podatke (u relativnom smislu).

Što dobivamo ovom interpretacijom?

- Onda možemo koristiti "normalna" pravila egzaktne 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

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

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

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

a za dijeljenje

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

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 Numeričku matematiku).

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 $\circ$ 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 $\varepsilon_\circ$ rezultata operacije $\circ$

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

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

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

Ako su $|\varepsilon_x|$ i $|\varepsilon_y|$ dovoljno mali da sve možemo linearizirati (tj. zanemariti "kvadratne" i više potencije epsilona), 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

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

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

$$(x \pm y)\,(1 + \varepsilon_\pm) := [\,x\,(1 + \varepsilon_x)\,] \pm [\,y\,(1 + \varepsilon_y)\,].$$

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

- Ako je i $x\,(1 + \varepsilon_x) \pm y\,(1 + \varepsilon_y) = 0$, relativna greška $\varepsilon_\pm$ može biti biti koji broj (nije određena), a prirodno je uzeti $\varepsilon_\pm = 0$.
- U protivnom, za $x\,(1 + \varepsilon_x) \pm y\,(1 + \varepsilon_y) \neq 0$, gornja jednakost je nemoguća, pa stavljamo $\varepsilon_\pm = \pm\infty$.

Širenje grešaka kod zbrajanja i oduzimanja

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

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

Relativnu grešku $\varepsilon_\pm$ 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,

tj. 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.

Nadalje, zbrajamo i oduzimamo brojeve istih predznaka.

Širenje grešaka kod zbrajanja

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

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

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

To vrijedi i kad je $x = 0$ ili $y = 0$. 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 ih 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, uz $x \neq 0$ i $y \neq 0$, sigurno vrijedi

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

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 to se može dogoditi i na oba mjesta.

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

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

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 mnogo 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 katastrofalno 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!

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

Primjer:
“Katastrofalno” kraćenje

Primjer katastrofalnog kraćenja

Zakruživanjem ulaznih podataka dolazi do male relativne greške. Kako ona može utjecati na konačni rezultat?

Primjer. Uzmimo realnu aritmetiku "računala" u bazi 10. Za mantisu (značajni dio broja) imamo $p = 4$ dekadske znamenke, a za eksponent imamo 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 $f\ell(x)$ i $f\ell(y)$, pravilno zaokružene na $p = 4$ znamenke

$$f\ell(x) = 8.887 \times 10^0,$$
$$f\ell(y) = 8.884 \times 10^0.$$

Primjer katastrofalnog kraćenja (nastavak)

Ovim zaokruživanjima napravili smo malu relativnu grešku u x i y (ovdje je $u = \frac{1}{2}\, b^{-p} = 5 \times 10^{-5}$).

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

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

Kod normalizacije, zbog pomaka "ulijevo", pojavljuju se

- ? = znamenke koje više ne možemo restaurirati (ta informacija se izgubila — zaokruživanjem x i y).

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 polazni operandi točni. Dakle, ovo oduzimanje je egzaktno i u aritmetici računala.

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

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 $f\ell(x) - f\ell(y)$ je pogrešna, a relativna greška je 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 $f\ell(x) - f\ell(y)$ bilo egzaktno i u aritmetici našeg "računala", ali rezultat je, svejedno, pogrešan.

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

- Uzrok su polazne greške u operandima $f\ell(x)$, $f\ell(y)$.

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

- i dalje, naravno, dolazi do kraćenja,
- ali je rezultat (uglavnom, a po IEEE standardu sigurno) egzaktan,

pa se ovo kraćenje onda zove benigno kraćenje.

Širenje grešaka u aritmetici računala

Dosad smo gledali širenje 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.

Ukratko, bez dokaza:

Svaka pojedina aritmetička operacija u računalu samo

- povećava perturbaciju svojih 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 — primjer

Primjer. Računamo zbroj $\hat{x} + \hat{y}$, gdje su $\hat{x}$ i $\hat{y}$ spremljeni u računalu i već imaju neku grešku, nastalu spremanjem pravih vrijednosti x i y u memoriju računala i eventualnim ranijim računom. Znamo da za izračunati rezultat vrijedi

$$fl(\hat{x} + \hat{y}) = (1 + \varepsilon_+)\,(\hat{x} + \hat{y}) = (1 + \varepsilon_+)\,\hat{x} + (1 + \varepsilon_+)\,\hat{y},$$

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

No, $\hat{x}$ i $\hat{y}$ već imaju neke relativne greške obzirom na prave egzaktne vrijednosti x i y

$$\hat{x} = (1 + \varepsilon_x)x, \quad \hat{y} = (1 + \varepsilon_y)y,$$

i to treba uvrstiti u gornju formulu.

Širenje grešaka u aritmetici računala — primjer

Dobivamo da je

$$\begin{aligned} f\ell(\hat{x} + \hat{y}) &= (1 + \varepsilon_+)\,(\hat{x} + \hat{y}) \\ &= (1 + \varepsilon_+)(1 + \varepsilon_x)\,x + (1 + \varepsilon_+)(1 + \varepsilon_y)\,y. \end{aligned}$$

Drugim riječima, izračunati rezultat $f\ell(\hat{x} + \hat{y})$ se opet može interpretirati kao egzaktni rezultat,

- ali na "malo više" perturbiranim polaznim podacima

$$\begin{aligned} \tilde{x} &= (1 + \eta_x)x = (1 + \varepsilon_+)(1 + \varepsilon_x)x, \\ \tilde{y} &= (1 + \eta_y)x = (1 + \varepsilon_+)(1 + \varepsilon_y)y. \end{aligned}$$

Relativne greške η_x i η_y su tzv. "greške unatrag" ili "obratne greške" — jer se odnose na perturbaciju ulaza (uz egzaktno računanje).

Širenje grešaka u aritmetici računala — primjer

Nažalost, prethodne formule ne daju neku informaciju o tome

- koliko je izračunati rezultat “daleko” od pravog rezultata,

a upravo to je ono što nas stvarno zanima.

- Ovo je tzv. “greška unaprijed”, jer mjeri “perturbaciju” rezultata, tj. izlaza (uz približno računanje).

Zadatak. Nađite relativnu grešku unaprijed η_+ u izračunatom rezultatu

$$fl(\hat{x} + \hat{y}) = (1 + \eta_+)\,(x + y),$$

u terminima ulaznih podataka x i y, te grešaka ε_x, ε_y i ε_+.

Zaključak. Greška unaprijed se puno teže računa od grešaka unatrag! Zato se obično nalazi “zaobilaznim” putem.

Natuknice o analizi grešaka

Bilo koji algoritam gledamo kao preslikavanje:

$$\text{ulaz (domena)} \;\to\; \text{izlaz (kodomena)}.$$

Naravno, zanima nas

- greška u izračunatom rezultatu — u kodomeni,
- uz približno računanje aritmetikom računala.

Ova greška zove se greška unaprijed (engl. forward error).

Nažalost, postupak "direktne" analize grešaka je težak,

- relativno rijetko "ide" i često daje loše ocjene greške.

Primjer. Obična norma vektora u $\mathbb{R}^2$ (i još dodaj "scaling").
(v. Z. Drmač, članak u MFL-u).

Natuknice o analizi grešaka (nastavak)

U praksi se puno češće koristi tzv. "obratna" analiza grešaka. Osnovna ideja je ista kao i za pojedine operacije:

- izračunati rezultati algoritma mogu se dobiti egzaktnim računanjem,
- ali na perturbiranim ulaznim podacima — u domeni.

Ova greška u domeni zove se greška unatrag ili obratna greška (engl. backward error).

Prednost: ocjena tih perturbacija u domeni je bitno lakša,

- jer se akumulacija onih faktora oblika $(1 + \varepsilon)$ prirodno radi unatrag — od rezultata prema polaznim podacima.

U protivnom, moramo znati grešku za operande, a to je greška unaprijed za prethodni dio algoritma.

Natuknice o analizi grešaka (nastavak)

Postupak "unatrag" za nalaženje grešaka u izračunatim rezultatima ide u dva koraka.

- Prvo se obratnom analizom naprave ocjene perturbacija polaznih podataka u domeni,
- a zatim se koristi matematička teorija perturbacije, koja daje ocjene grešaka rezultata u kodomeni. Ovaj izvod ide za egzaktni račun, pa vrijede sva normalna pravila.

Tako stižemo do pojmova:

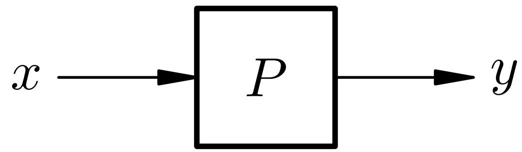
- stabilno i nestabilno računanje ili algoritam $=$ "prigušivač" ili "pojačalo" grešaka.

Slikice (skripta NA, Higham) — su na sljedećoj stranici.

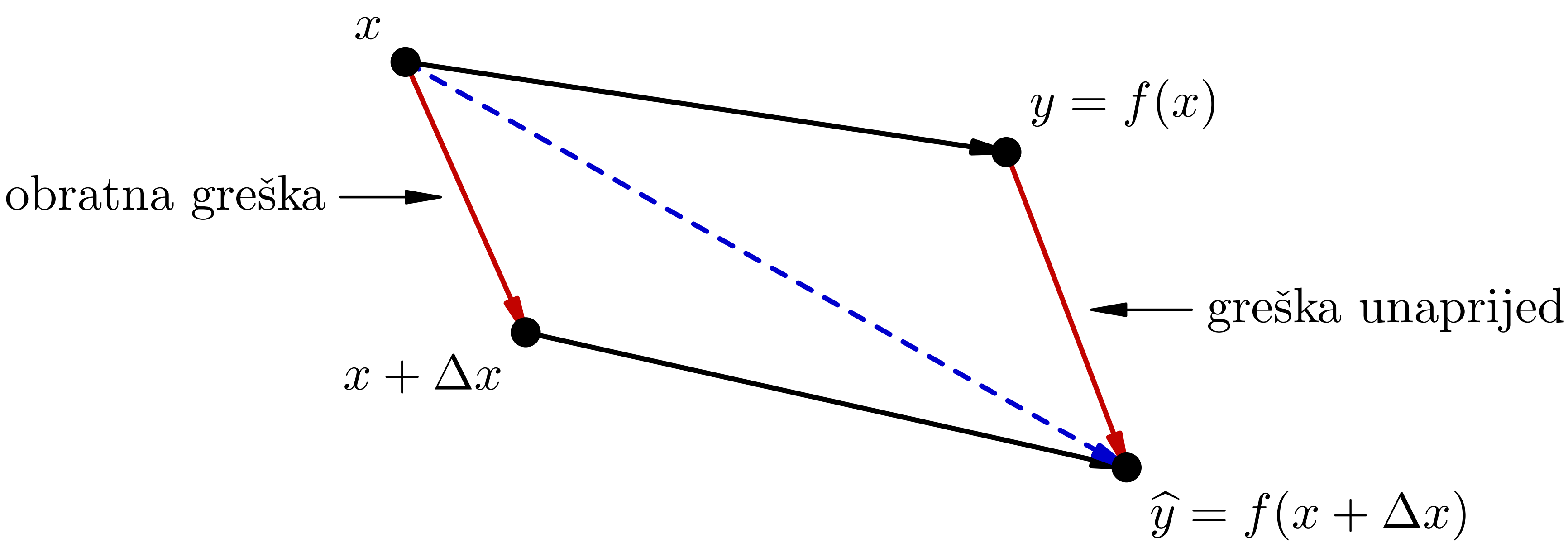
- Primjeri nestabilnosti — uklonjivi i NEuklonjivi.

Greška unaprijed i obratna greška

Uzmimo da algoritam “rješava” problem P.



Ako problem P intepretiramo kao računanje funkcije f, onda grešku unaprijed i obratnu grešku možemo prikazati ovako:



Dodatna literatura za floating point aritmetiku

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

- 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.

Primjer:
Kvadratna jednadžba

Kvadratna jednadžba

Uzmimo da treba riješiti (realnu) kvadratnu jednadžbu

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

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

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 jednadžba — standardni oblik

Za početak, jer znamo da je $a \neq 0$, onda jednadžbu možemo podijeliti s a, tako da dobijemo tzv. "normalizirani" oblik

$$x^2 + px + q = 0, \qquad p = \frac{b}{a}, \quad q = \frac{c}{a}.$$

Po standardnim formulama, rješenja ove jednadžbe su

$$x_{1,2} = \frac{-p \pm \sqrt{p^2 - 4q}}{2}.$$

Međutim, u praksi, stvarno računanje se radi po formuli

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q},$$

s tim da na početku izračunamo i zapamtimo $p/2$ ili $-p/2$. Ovim postupkom štedimo jedno množenje (ono s 4).

Kvadratna jednadžba — problem

Primjer. Rješavamo kvadratnu jednadžbu $x^2 - 56x + 1 = 0$.

U dekadskoj aritmetici s $p = 5$ značajnih znamenki dobijemo

$$x_1 = 28 - \sqrt{783} = 28 - 27.982 = 0.018000,$$

$$x_2 = 28 + \sqrt{783} = 28 + 27.982 = 55.982.$$

Točna rješenja su

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

Apsolutno manji od ova dva korijena — x_1, ima samo dvije točne znamenke (kraćenje), relativna greška je $7.7 \cdot 10^{-3}$!

Apsolutno veći korijen x_2 je “savršeno” točan.

Kvadratna jednadžba — popravak

Prvo izračunamo većeg po apsolutnoj vrijednosti, po formuli

$$x_2 = \frac{-(b + \operatorname{sign}(b)\sqrt{b^2 - 4ac})}{2a} = -\frac{p}{2} - \operatorname{sign}(p)\sqrt{\left(\frac{p}{2}\right)^2 - q},$$

a manjeg po apsolutnoj vrijednosti, izračunamo iz

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

(Vièteova formula), tj. formula za x_1 je

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

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 s velikim kraćenjem — nema jednostavnog rješenja. Naime, "krivac" nije aritmetika.
    - To je samo odraz tzv. nestabilnosti problema. Tad imamo dva bliska korijena, koji su vrlo osjetljivi na male promjene (perturbacije) koeficijenata jednadžbe.
    - Na primjer, pomak c = pomak grafa "gore–dolje". Mali pomak rezultira velikom promjenom korijena!

Primjeri izbjegavanja kraćenja

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!

Širenje grešaka — zaključak

Opasne i bezopasne operacije — sažetak

Jedina opasna operacija — kad rezultat može imati veliku relativnu grešku, je

- oduzimanje bliskih brojeva,
- i to samo kad polazni operandi već imaju neku grešku (samo oduzimanje je tada, najčešće, egzaktno).

Ovaj fenomen zove se opasno ili katastrofalno kraćenje.

Sve ostale operacije su bezopasne — relativna greška rezultata ne raste pretjerano. Posebno,

- dijeljenje malim brojem nije opasno,
- osim kad je mali broj nastao (ranijim) kraćenjem.

Nažalost, u nekim knjigama piše suprotno — i pogrešno.