Može li se vjerovati računalu?

Sanja i Saša Singer

Tri najtvrdokornije fame o računalima

O računalima postoje tvrdo ukorijenjene fame kao što postoje i za ljude, na primjer o škrtosti Škota, nemarnosti Francuza ili ljubavnim podvizima Talijana. Fame o računalima koje je vrlo teško razbiti su:

- računalom se sve može izračunati,
- rješenje se uvijek dobiva u kratkom vremenu,
- računalo uvijek daje točne rezultate.

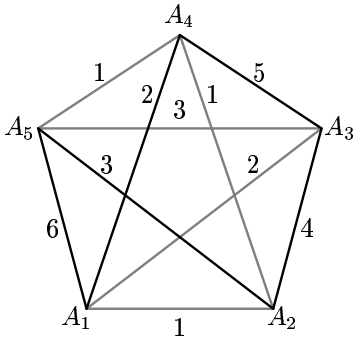
Međutim, niti jedna od te tri tvrdnje nije istinita! Prva od njih je vjerojatno najlakše objašnjiva. Računala su prilagođena **ponavljanju** radnji, pa ako u nizu operacija nema ponavljanja, onda takav postupak nije primjeren za računalo.

Druga je tvrdnja mnogo suptilnija. U većini situacija, moderna računala "izbacuju" rezultate i prije no što trepnemo okom, ali postoje realni problemi kojima nije moguće dočekati završetak.

Najpoznatiji problem koji ilustrira dugotrajnost rješavanja je problem trgovačkog putnika.

Primjer 1. Zadano je n gradova, tako da su svaka dva vezana cestom. Također, zadane su cijene putovanja. Trgovački putnik kreće iz grada A_1, obilazi sve ostale gradove samo jednom i vraća se ponovno u A_1 (zatvori ciklus). Ako gradovi nemaju cestu koja ih direktno povezuje, za cijenu puta između ta dva grada možemo staviti ∞, odnosno, u praktičnoj realizaciji, neki dovoljno veliki broj. Cilj trgovačkog putnika je naći ciklus najmanje cijene.

Na primjer, za problem trgovačkog putnika



lako je provjeriti da je ciklus najmanje cijene $(A_1, A_3, A_5, A_4, A_2, A_1)$ ili, naravno, obratan ciklus $(A_1, A_2, A_4, A_5, A_3, A_1)$, i da mu je cijena 8 jedinica.

Algoritam za egzaktno rješavanje ovog problema je ispitivanje svih ciklusa, a njih ima $(n-1)!$. Objašnjenje za broj ciklusa je jednostavno. Ako krećemo

iz grada A_1, drugi grad u koji stižemo možemo odabrati na $n-1$ načina, treći grad možemo izabrati na $n-2$ načina (jer se ne smijemo vratiti u početni grad ili ostati u A_2), ...

Računanje cijene odgovarajućeg ciklusa zahtijeva n zbrajanja. Prema tome, za rješenje problema trgovačkog putnika potrebno je približno $n!$ računskih operacija.

Izračunajmo koliko bi trajalo egzaktno rješavanje problema trgovačkog putnika za 10, 20, 30, 40 i 50 gradova. Pretpostavimo da nam je na raspolaganju najmodernije PC računalo koje izvodi reda veličine 10^8 računskih operacija u sekundi.

n	sekundi	sati	dana	godina
10	$3.6288 \cdot 10^{-02}$	$1.0080 \cdot 10^{-05}$	$4.2000 \cdot 10^{-07}$	$1.1507 \cdot 10^{-09}$
20	$2.4329 \cdot 10^{+10}$	$6.7581 \cdot 10^{+06}$	$2.8159 \cdot 10^{+05}$	$7.7147 \cdot 10^{+02}$
30	$2.6525 \cdot 10^{+24}$	$7.3681 \cdot 10^{+20}$	$3.0701 \cdot 10^{+19}$	$8.4111 \cdot 10^{+16}$
40	$8.1592 \cdot 10^{+39}$	$2.2664 \cdot 10^{+36}$	$9.4435 \cdot 10^{+34}$	$2.5873 \cdot 10^{+32}$
50	$3.0414 \cdot 10^{+56}$	$8.4484 \cdot 10^{+52}$	$3.5201 \cdot 10^{+51}$	$9.6442 \cdot 10^{+48}$

Uočite da, osim egzaktnog rješenja problema za 10 gradova, ostali problemi nisu rješivi u razumnom vremenu, jer već za $n = 20$, za rješenje problema treba čekati 771 godinu!

Moderna znanost pretpostavlja da je Zemlja stara oko 4.5 milijarde godina (tj. $4.5 \cdot 10^9$ godina), a rješavanje problema za $n = 30$ gradova trajalo bi sedam redova veličine dulje.

Kad bismo na raspolaganju imali neko od danas najbržih, paralelnih računala, koje izvodi približno 10^{13} računskih operacija u sekundi, brojke u prethodnoj tablici bile bi 10^5 puta manje. U tom slučaju, jedino još kako-tako smisleno bilo bi čekati 2.8 dana za rješenje problema s 20 gradova.

Što nam prethodni primjer pokazuje? Pokazuje da ne bismo trebali problem trgovačkog putnika rješavati egzaktno, ali nipošto ne kaže da ga uopće ne bismo trebali rješavati! Postoje algoritmi koji dobro (i relativno brzo) približno rješavaju ovaj problem.

Koja je posebnost egzaktnog algoritma za rješavanje problema trgovačkog putnika? Naime, ako imamo problem dimenzije n (tj. n gradova), vrijeme traženja rješenja (ili broj potrebnih aritmetičkih operacija) **eksponencijalno raste** u ovisnosti o n, što slijedi iz nejednakosti

$$n^{n/2} \leq n! \leq n^n.$$

□

Ako je prethodna dva objašnjenja možda i moguće jednostavno prihvatiti, ono posljednje o tome da računalo griješi ljudi obično ne žele prihvatiti.

Vjerovati ili ne?

Većina ljudi bi na ovo pitanje bez oklijevanja odgovorila potvrdno. Kako možemo ne vjerovati toj sjajnoj spravi, dometu moderne tehnologije? Zašto su ga ljudi izmislili, ako sumnjaju u njega i ako sve njegove rezultate moramo provjeravati. Računalo ne može pogriješiti jer ga danas boli glava, ili zato što je zbog previše posla zaboravilo na neki podatak! Sve što ono izračuna je sigurno potpuno točno i tim rezultatima treba bezrezervno vjerovati. Ako je netko pogriješio, sigurno je pogriješio čovjek koji je pisao programe, a ne računalo!

Nažalost, ne bismo se složili s tim mišljenjem. Ako računalo koristite kao pisaći stroj za uređivanje izgleda teksta, onda je vjerojatno istina da je za eventualnu grešku, koja ne dozvoljava neku opciju za uljepšavanje teksta, kriv pisac programa za uređivanje teksta, a ne računalo.

Ali, ako računalo koristite za računanje, oprezno s dobivenim rezultatima. Postoji mogućnost da je čovjek–pisac prevoditelja (kompajlera) negdje pogriješio, pa zbog njegove greške nešto ne radi. Ili, krivac možete biti Vi sami, jer ste program neispravno napisali, pa računalo ima pogrešne naredbe. Ili, krivac može biti proizvođač procesora koji je neispravno napravio procesor. Ova posljednja mogućnost je najmanje vjerojatna i po našim saznanjima, od nje je patila prva serija Pentium 90 procesora koja je kod dijeljenja nekih brojeva (vrlo malog skupa njih) umjesto 18 točnih decimalnih znamenki davala 6.

Na kraju, postoji i četvrta mogućnost zašto računalo griješi. Naprosto zato jer je njegova aritmetika napravljena tako da mora malo griješiti kod računanja s realnim brojevima. Pitanje je samo može li se dogoditi da male greške u svakoj operaciji kulminiraju velikim greškama u rezultatu. Dakle, ako isključimo ljudske pogreške, pokažimo da pogreške aritmetike računala mogu biti presudne za točnost rezultata. Pokazalo se da zbog tih pogrešaka može doći i do ljudskih žrtava i do velike materijalne štete.

Izvori grešaka

Izvori grešaka u rješavanju realnih problema su:

- model,
- metoda za rješavanje modela,
- ulazni podaci (mjerenja),
- aritmetika računala.

Sve četiri vrste grešaka lako je razumjeti. Međutim, za posljednju, vrlo je teško vjerovati da ona može biti toliko značajna—dominantna u odnosu na ostale, tako da je rezultat zbog nje besmislen.

Aritmetika računala

Što zaista moramo znati o aritmetici računala da bismo imali elementarnu podlogu za vjerovanje rezultatima? Moramo znati:

- način prikaza brojeva u računalu,
- porijeklo grešaka zaokruživanja,
- osnovna pravila za greške (jer one nisu slučajne).

U računalu postoje **dva** bitno različita tipa (skupa) brojeva:

- "cijeli" brojevi—`integer`,
- "realni" brojevi—`real`.

Osnovna razlika od "matematičkih" skupova $\mathbb{Z}$ i $\mathbb{R}$ je da su oba skupa **konačni** skupovi, tj. pravi podskupovi od $\mathbb{Z}$, odnosno $\mathbb{R}$.

Prva posljedica te činjenice je da u oba tipa postoje brojevi koje **ne možemo** prikazati u računalu! Zajedničko svojstvo oba tipa je da se za prikaz brojeva koristi pozicioni zapis u bazi 2.

Cjelobrojna aritmetika

Osnovna svojstva prikaza i aritmetike cijelih brojeva su:

- duljina ℓ bitova, ℓ fiksan,
- aritmetika je modularna aritmetika u prstenu ostataka modulo 2^ℓ,
- sistem ostataka je simetričan oko 0 (tako da je prvi bit predznak).

Prema tome, prikazivi brojevi su:

$$-2^{\ell-1}, \ldots, -1, 0, 1, \ldots, 2^{\ell-1} - 1,$$

pri čemu su tipične vrijednosti za ℓ: 16, 32 i 64.

S cjelobrojnom aritmetikom nema velikih problema, ako su razultati prikazivi, onda su oni i točni. Što ako nisu prikazivi, tj. ako su preveliki? Budući da je aritmetika cijelih brojeva modularna, računalo neće javiti grešku, već će "modularno" izračunati rezultat.

Na primjer, za $\ell = 32$ je

$$\begin{aligned} \text{najveći broj} &= \quad 2^{31} - 1 = \quad 2\,147\,483\,647 \\ \text{najmanji broj} &= -2^{31} \qquad = -2\,147\,483\,648, \end{aligned}$$

pa je

$$2^{31} - 1 + 1 = -2^{31}.$$

Prethodni primjer pokazuje što je "nezdravo" računati u cjelobrojnoj aritmetici. To su svi rezultati koji brzo rastu (ili padaju). Na primjer, iako je $n!$

cijeli broj, već za razumno mali n, postaje prevelik za cjelobrojnu aritmetiku, pa je rezultat bolje čuvati kao realan broj, inače bi nam se moglo dogoditi da rezultati "plešu". Usporedimo rezultate za $n!$ dobivene u cjelobrojnoj aritmetici sa $\ell = 16$ i $\ell = 32$ bita s rezultatom (ispravnim) dobivenim u realnoj aritmetici.

n	$\ell = 16$	$\ell = 32$	realna aritmetika
7!	5040	5040	5040
8!	−25216	40320	40320
9!	−30336	362880	362880
10!	24320	3628800	3628800
11!	5376	39916800	39916800
12!	−1024	479001600	479001600
13!	−13312	1932053504	6227020800
14!	10240	1278945280	87178291200
15!	22528	2004310016	1307674368000
16!	−32768	2004189184	20922789888000
17!	−32768	−288522240	355687428096000
18!	0	−898433024	6402373705728000
19!	0	109641728	121645100408832000
20!	0	−2102132736	2432902008176640000

Ako rezultate napišemo ovako, redom, lako ćemo uočiti gdje se cjelobrojna aritmetika počinje "modularno" ponašati. Ali, ako računamo, na primjer, samo 13! u 32-bitnoj cjelobrojnoj aritmetici, nismo odmah u stanju otkriti da je rezultat pogrešan (približno je dobrog reda veličine i pozitivan).

Zadatak 1. Nađite najmanji broj n, takav da je

$$n! = 0$$

u cjelobrojnoj aritmetici duljine $\ell = 32$ bita. Postoji li uvijek takav n za proizvoljnu duljinu mantise ℓ? □

Zadatak 2. Nađite n za koji je Fibonaccijev broj f_n, $n \in \mathbb{N}_0$,

$$f_n = f_{n-1} + f_{n-2}, \quad f_0 = 0, \quad f_1 = 1,$$

prevelik za cjelobrojnu aritmetiku s $\ell = 16$ i $\ell = 32$ bita. □

Realna aritmetika

Realni brojevi prikazuju se korištenjem mantise m i eksponenta e:

$$r \pm m \cdot 2^e,$$

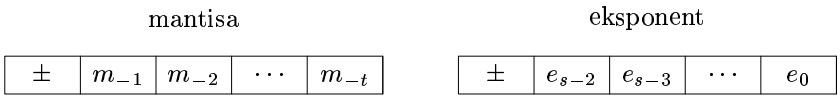
pri čemu je

- e cijeli broj u određenom rasponu,
- m racionalni broj, $1/2 \leq m < 1$,
- dogovorno za $r = 0$ je $e = 0$, $m = 0$.

Osim toga, zbog konačnosti prikaza:

- eksponent e je s-bitni cijeli broj,
- za mantisu m pamti se prvih t znamenki iza binarne točke.

To izgleda ovako:



Skup realnih brojeva **prikazivih** u računalu parametriziran je duljinom mantise t i duljinom eksponenta s, u oznaci $\mathbb{R}(t, s)$.

Primijetite da postoje realni brojevi koje ne možemo egzaktno spremiti u računalo, čak i kad su unutar prikazivog raspona brojeva. Neka je $x \in \mathbb{R}$ unutar prikazivog raspona i

$$x = \pm \left(\sum_{k=1}^{\infty} b_{-k} 2^{-k} \right) 2^e.$$

Ako mantisa od x ima više od t znamenki, sprema se (najbliža) aproksimacija $fl(x) \in \mathbb{R}(t, s)$ koja se može prikazati kao

$$fl(x) = \pm \left(\sum_{k=1}^{t} b^*_{-k} 2^{-k} \right) 2^{e^*}.$$

Broj $fl(x)$ dobivamo **zaokruživanjem** broja x:

- prva odbačena znamenka 1 – zaokružuje se nagore,
- prva odbačena znamenka 0 – zaokružuje se nadolje.

Time smo napravili apsolutnu grešku zaokruživanja manju ili jednaku polovini vrijednosti zadnjeg bita, tj.

$$|x - fl(x)| \leq 2^{-t-1+e}.$$

Zgodnije je ocijeniti relativnu grešku

$$\left|\frac{x - fl(x)}{x}\right| \leq \frac{2^{-t-1+e}}{2^{-1} \cdot 2^{e}} = 2^{-t}.$$

Dakle, imamo vrlo malu relativnu grešku. U oznaci, jedinična greška zaokruživanja (engl. unit roundoff) je

$$u := 2^{-t}.$$

Ako je $x \in \mathbb{R}$ unutar raspona brojeva prikazivih u računalu, onda se umjesto x sprema zaokruženi broj $fl(x) \in \mathbb{R}(t, s)$ i vrijedi

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

gdje je ε relativna greška napravljena tim zaokruživanjem.

Da svako računalo ne bi imalo svoje duljine mantisa i eksponenata, te aritmetiku koja se ponaša "kako hoće", postoji standard koji to propisuje, tzv. IEEE standard za realne tipove:

	`single`	`double`	`extended`
duljina	32 bita	64 bita	80 bitova
mantisa	$23 + 1$ bit	$52 + 1$ bit	64 bita
eksponent	8 bitova	11 bitova	15 bitova
u	2^{-24}	2^{-53}	2^{-64}
$u \approx$	$5.96 \cdot 10^{-8}$	$1.11 \cdot 10^{-16}$	$5.42 \cdot 10^{-20}$
raspon $\approx$	$10^{\pm 38}$	$10^{\pm 308}$	$10^{\pm 4932}$

Za sva tri tipa rezerviran je još jedan bit za predznak. Kod tipova `single` i `double` dodatni bit u duljini mantise je tzv. "sakriveni bit" (engl. hidden bit) jer je prvi znak iza binarne točke uvijek 1, pa se ne mora pamtiti.

Aritmetičke operacije se mogu izvesti samo na operandima koji su već spremljeni u memoriji, dakle pripadaju skupu $\mathbb{R}(t, s)$.

Osnovna pretpostavka je da za osnovne aritmetičke operacije vrijedi ista ocjena greške zaokruživanja kao i za prikaz brojeva. Preciznije, neka $\circ$ označava bilo koju operaciju $+$, $-$, $*$, $/$. Onda za $x, y \in \mathbb{R}(t, s)$ vrijedi

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

za sve $x, y \in \mathbb{R}(t, s)$ za koje je $x \circ y$ u dozvoljenom rasponu. Ova ocjena je ekvivalentna idealnom izvođenju operacija:

- egzaktno se izračuna rezultat operacije $x \circ y$,
- rezultat se zaokruži pri spremanju u memoriju!

Kad imamo puno aritmetičkih operacija, očito dolazi do akumulacije grešaka zaokruživanja. Ključno je pitanje možemo li išta reći o tom "rasprostiranju" grešaka zaokruživanja? Na sreću, možemo!

Nažalost, aritmetika računala nije egzaktna i u njoj ne vrijede uobičajeni zakoni za operacije. Na primjer, za zbrajanje i množenje nema asocijativnosti i distributivnosti.

Primjer 2. Zbrajanje brojeva računalom nije asocijativno. Izračunajmo

$$S_1 = \sum_{i=1}^{1000000} \frac{1}{i}, \qquad S_2 = \sum_{i=1000000}^{1} \frac{1}{i}$$

u tri IEEE točnosti. Dobiveni rezultati su:

	`single`	`double`	`extended`
S_1	14.3573579788208008	14.3927267228647810	14.3927267228657234
S_2	14.3926515579223633	14.3927267228657545	14.3927267228657236

Primijetite da nešto točniji rezultat daje zbrajanje S_2. Razlog leži u činjenici da zbrajamo od manjih brojeva prema većim, pa se zbroj pomalo "nakuplja". Napravimo li obratno, tj. zbrajamo li od velikih brojevima prema manjim, ako je zbroj S dovoljno velik, onda mali dodani član ne utječe na rezultat, pa zbroj ostaje isti. □

Analiza pojedinih operacija postaje bitno lakša, ako uočimo da greške zaokruživanja možemo interpretirati i kao **egzaktne** operacije, ali na "malo" pogrešnim podacima! Dovoljno je $1+\varepsilon$ u ocjeni greške "zalijepiti" za x i/ili y. To je isto kao da operand(i) ima(ju) grešku na ulazu u operaciju, a operacija $\circ$ je egzaktna. Onda možemo koristiti "normalna" pravila aritmetike za analizu grešaka.

Pretpostavimo onda da su podaci x i y malo perturbirani, s relativnim greškama

$$|\varepsilon_x|, |\varepsilon_y| \leq u.$$

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

Množenje je dobroćudna operacija:

$$x(1+\varepsilon_x) * y\,(1+\varepsilon_y) \approx xy\,(1+\varepsilon_x+\varepsilon_y) := xy(1+\varepsilon_*),$$

uz ocjenu relativne greške

$$|\varepsilon_*| \leq 2u.$$

Greška se samo zbraja.

Dijeljenje je dobroćudna operacija:

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

uz istu ocjenu relativne greške

$$|\varepsilon_/| \leq 2u.$$

Greška se opet samo zbraja.

Neka su x i y proizvoljnog predznaka. Za zbrajanje (oduzimanje) vrijedi:

$$x\,(1+\varepsilon_x)+y\,(1+\varepsilon_y)=(x+y)\Big(1+\frac{x\varepsilon_x+y\varepsilon_y}{x+y}\Big),$$

ako je $x+y\neq 0$, pa definiramo

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

Zbrajanje brojeva istog predznaka je dobroćudna operacija.
Tada vrijede ocjene

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

pa je $|\varepsilon_\pm|\leq 2u$. Greška se opet samo zbraja.

Zbrajanje brojeva različitog predznaka je opasna operacija.
Ako je $|x+y|\ll|x|,|y|$, kvocijenti

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

mogu biti proizvoljno veliki, pa i relativna greška $|\varepsilon_\pm|$ rezultata može biti proizvoljno velika! Opasnost nastaje kad je rezultat zbrajanja brojeva suprotnog predznaka broj koji je po apsolutnoj vrijednosti mnogo manji od polaznih podataka, tj. kad nastupi tzv. opasno kraćenje.

Primjer 3. Radi jednostavnosti, pretpostavimo da na raspolaganju imamo dekadsko računalo (računalo s bazom 10) kojemu je mantisa duga $t=4$ znamenke, a eksponent $s=2$ znamenke. Na takvom računalu od broja x oduzmimo broj y, ako je

$$x=0.99966=0.99966\cdot 10^0,\quad y=0.99944=0.99944\cdot 10^0.$$

Umjesto brojeva x i y koji imaju predugu mantisu, spremljeni su brojevi

$$fl(x)=0.9997\cdot 10^0,\quad fl(y)=0.9994\cdot 10^0$$

čime je napravljena mala relativna greška. Oduzimamo znamenku po znamenku mantise, pa normaliziramo

$$0.9997\cdot 10^0-0.9994\cdot 10^0=0.0003\cdot 10^0=0.3???\cdot 10^{-3}.$$

Znakom upitnika označene su znamenke koje više ne možemo restaurirati, pa računalo na ta mjesta upisuje 0. Pravi rezultat je $0.22\cdot 10^{-3}$, pa je već prva značajna znamenka pogrešna! Primijetite da je samo oduzimanje bilo egzaktno

za $fl(x)$ i $fl(y)$, ali rezultat je pogrešan. Na prvi pogled čini nam se da znamo bar red veličine rezultata i da to nije tako strašno. Prava katastrofa nastupa ako $0.3??? \cdot 10^{-3}$ uđe u naredna zbrajanja i oduzimanja i ako se pritom "skrati" i ta trojka. Tada nemamo nikakve kontrole nad rezultatom. □

Primjer 4. Računalom treba naći korijene realne kvadratne jednadžbe

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

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

Matematički gledano, problem je trivijalan: imamo 2 rješenja

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

Numerički gledano, problem je mnogo izazovniji. Koliko su stvarno točni korijeni izračunati po ovoj formuli?

Na primjer, za

$$x^2 - 56x + 1 = 0$$

u aritmetici s 5 dekadskih znamenki dobivamo

$$x_1 = \frac{56 - \sqrt{3132}}{2} = \frac{56 - 55.964}{2} = 0.018000,$$
$$x_2 = \frac{56 + \sqrt{3132}}{2} = \frac{56 + 55.964}{2} = 55.982.$$

Točna rješenja su

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

Manji od ova dva korijena ima samo dvije točne znamenke, jer su se ostale izgubile kraćenjem.

Možemo li ikako izbjeći kraćenje? Možemo! Prvo izračunamo po apsolutnoj vrijednosti veći korijen, po formuli

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

a zatim manji iz Vietaine formule $ax_1 \cdot x_2 = c$, tj. iz

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

pa je opasno kraćenje zamijenjeno dobroćudnim dijeljenjem! □

Primjer 5. Računalom koje koristi dekadsku aritmetiku s 5 značajnih znamenki, treba naći korijene realne kvadratne jednadžbe

$$4x^2 + 4.0001x + 1 = 0.$$

Već kod računanja diskriminante nastaje kraćenje, jer je u dekadskoj aritmetici s 5 značajnih znamenki

$$4.0001^2 - 4^2 = 0.$$

Zbog toga, kao rješenje dobivamo dvostruki korijen

$$x_{1,2} = -0.5,$$

umjesto pravog rješenja

$$x_1 = -0.496476944\ldots \quad \text{i} \quad x_2 = -0.503548056\ldots.$$

Nažalost, za ovaj slučaj nema jednostavnog rješenja koje bi spriječilo katastrofalno kraćenje kod računanja diskriminante. □

Primjer 6. Vrijednost

$$f_n(x) = (x-n)^{10}, \quad n \in \mathbb{N}_0,$$

računamo u aritmetici računala u okolini točke n. Primijetite da je graf funkcije $(x-n)^{10}$ translatirani graf funkcije x^{10} za n jedinica udesno.

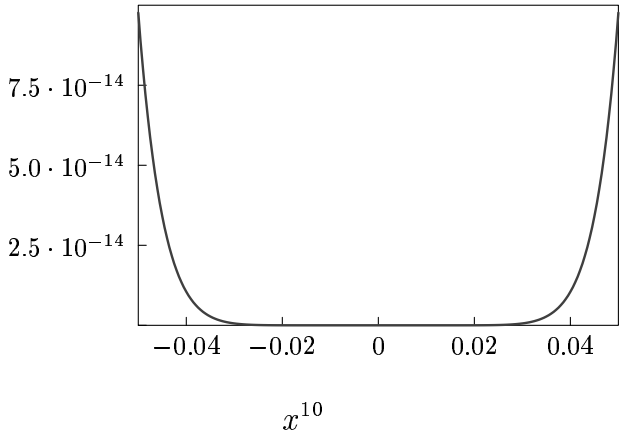
Funkcijsku vrijednost funkcija f_n možemo izračunati na više načina koji su matematički ekvivalentni, ali nisu numerički jednaki. Recimo, možemo translatirati graf funkcije x^{10} za n jedinica udesno, ili možemo koristiti binomnu formulu

$$(x-n)^{10} = \sum_{k=0}^{10} \binom{10}{k} (-n)^{10-k} x^k,$$

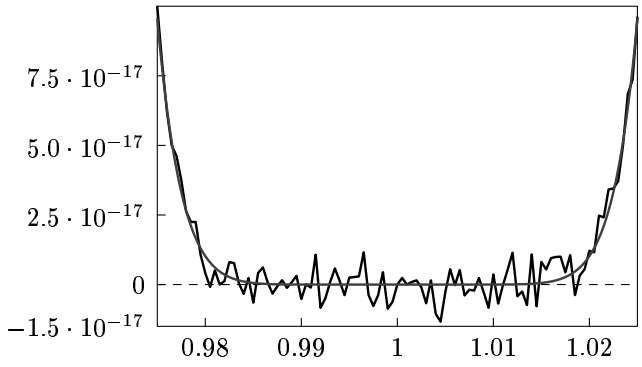
s tim da polinom na desnoj strani računamo Hornerovom shemom. Što je točnije?

U okolini točke n je $(x-n)^{10}$ mali broj. Pogledajmo kakvi su članovi u sumi na desnoj strani. Koeficijenti s desne strane su **alternirajući** po predznaku i **rastu** s porastom n. U sumi mora doći do kraćenja, pa je rezultat bolje izračunati direktno. Čak se može i predvidjeti red veličine greške, znajući veličinu koeficijenata koji se javljaju u sumi.

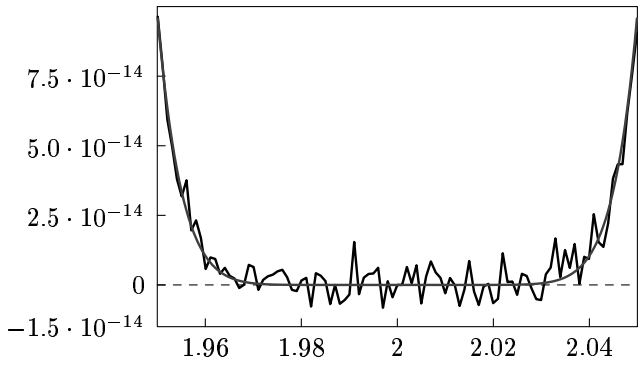
Evo kako izgledaju grafovi dobiveni translacijom i korištenjem binomne formule.



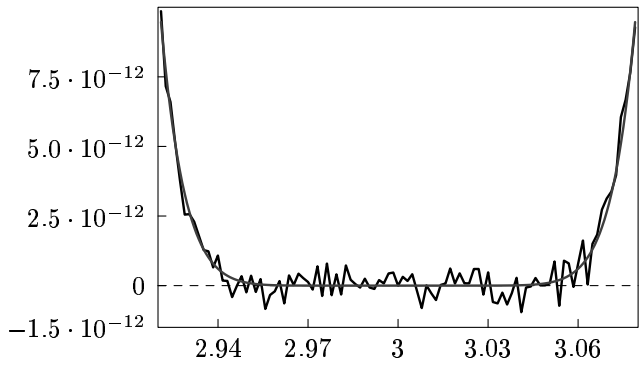
x^{10}



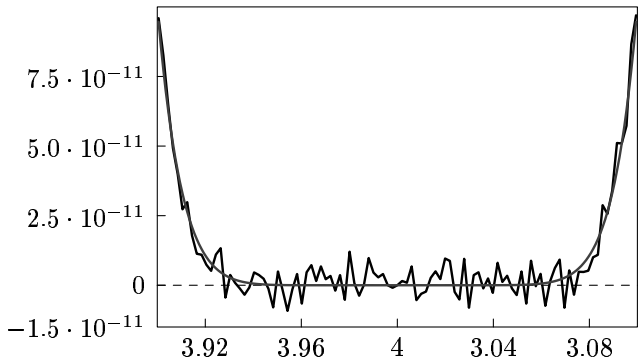
$$(x-1)^{10} = x^{10} - 10x^9 + 45x^8 - 120x^7 + 210x^6 - 252x^5 + 210x^4 - 120x^3 + 45x^2 - 10x + 1.$$



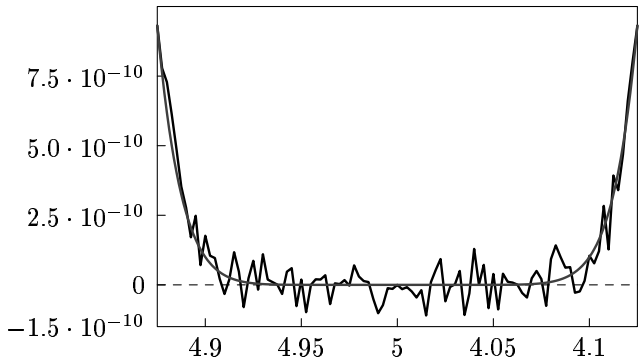
$$(x-2)^{10} = x^{10} - 20x^9 + 180x^8 - 960x^7 + 3360x^6 - 8064x^5 + 13440x^4 - 15360x^3 + 11520x^2 - 5120x + 1024.$$



$$(x-3)^{10} = x^{10} - 30x^9 + 405x^8 - 3240x^7 + 17010x^6 - 61236x^5 + 153090x^4 - 262440x^3 + 295245x^2 - 196830x + 59049.$$



$$(x-4)^{10} = x^{10} - 40x^9 + 720x^8 - 7680x^7 + 53760x^6 - 258048x^5 + 860160x^4 - 1966080x^3 + 2949120x^2 - 2621440x + 1048576.$$



$$(x-5)^{10} = x^{10} - 50x^9 + 1125x^8 - 15000x^7 + 131250x^6 - 787500x^5 + 3281250x^4 - 9375000x^3 + 17578125x^2 - 19531250x + 9765625.$$

Zadatak 3. Broj 2.001^{10} aproksimiramo na računalu, korištenjem MacLaurinovog reda za funkciju

$$f(x) = (x-2)^{10}.$$

Hoće li takva aproksimacija biti približno točna ili ne? Koje dvije vrijednosti možete staviti za x? Koja će od njih dati točniji rezultat? □

Zadatak 4. Broj $\operatorname{ctg}(75\pi/4)$ računamo putem početnih komada MacLaurinovih redova za funkcije $\cos x$ i $\sin x$ koristeći aritmetiku računala, sve dok posljednji član u svakom od zbrojeva ne padne ispod zadane točnosti ε ($0 < \varepsilon \ll 1$). Objasnite zašto će takva aproksimacija dati netočan rezultat! Možete li nekom transformacijom argumenta funkcije osigurati točnost rezultata? □

Zaključak

Ponašanje rezultata se može predvidjeti — strah je nepotreban, a oprez nužan. Imamo li sumnju da je oduzimanjem brojeva došlo do kraćenja, svakako treba pokušati problem preformulirati ili provjeriti drugom metodom i/ili u višoj točnosti. Jasno je da treba poznavati i greške metode.