

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

Sanja Singer

Numerička matematika

Predavanja na FSB

Zagreb, 2004/5.

1. Mogućnosti današnjih računala

Možda nije najsretnije rješenje početi pričati o mogućnosti današnjih računala, kad se zna kojom se brzinom mijenjaju i ubrzavaju. Ipak, neke osnovne postavke ostat će nepromijenjene, bez obzira poveća li se broj osnovnih aritmetičkih operacija u sekundi koje računalo može izvoditi.

Često, ali pogrešno je mišljenje da se računalom mogu rješavati svi problemi — podaci se “ubace” u računalo, a ono nakon nekog vremena izbací točan rezultat. Zaboravlja se na činjenicu da današnja računala nisu “inteligentna” (iako se tomu teži) i da su svi procesi u računalu vođeni ljudskom rukom.

Što se onda ipak može riješiti računalom? Mogu se riješiti svi problemi za koje postoji točno definiran, konačan postupak rješavanja — algoritam.

Što je algoritam? Prema definiciji Donalda Knutha, uvaženog stručnjaka za računarstvo, algoritam je konačan niz operacija koji rješava neki problem. Osim toga, algoritam mora zadovoljavati još i:

1. konačnost — za svaki ulaz, algoritam mora završiti nakon konačnog broja koraka;
2. točnu definiranost — u svakom koraku algoritma točno se zna sljedeći korak (nema slučajnosti);
3. algoritam može, ali i ne mora, imati ulazne podatke;
4. algoritam **mora** imati izlazne podatke;
5. efikasnost — svaki algoritam mora završiti u razumnom vremenu.

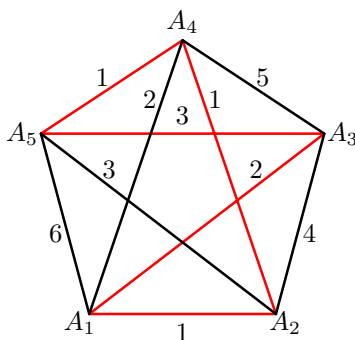
Možda je potrebno prokomentirati samo posljednja tri zahtjeva. Uobičajeno je da algoritmi imaju ulazne podatke, ali to nije nužno. Na primjer, ako želimo izračunati konstantu π nekim algoritmom, takav program vjerojatno neće zahtijevati ulazne podatke. Sasvim je obrnuta situacija s izlaznim podacima. Ako ih nema, algoritam je obavio neki posao za nas, ali nas nije izvjestio o krajnjem rezultatu. A što mi znamo o rješenju? Jednako kao da algoritam nije obavio nikakav posao!

1.1. Efikasni i neefikasni algoritmi

Od svih ovih zahtjeva koji se postavljaju na algoritam, najnerazumljiviji je zahtjev efikasnosti. Zahtjev efikasnosti znači da rješenje problema treba pronaći u razumnom vremenu.

Primjer 1.1.1. (Problem trgovačkog putnika) *Neka je zadano 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 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 najmoder-

nije 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

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

Desna nejednakost je trivijalna. Dakle, ostaje nam pokazati samo prvu. Ako pokažemo da vrijedi

$$k(n - k + 1) \geq n, \quad k \in \{1, \dots, n\}, \quad (1.1.1)$$

onda produkt $1 \cdot 2 \cdots n$ možemo organizirati tako da množimo prvi i posljednji član, drugi i pretposljednji, i tako redom. Takvih parova produkata ima $n/2$, pa je

rezultat očit. Dakle, preostaje pokazati samo relaciju (1.1.1). Prebacivanjem člana n na lijevu stranu slijedi

$$(k - 1)n - k^2 + k = (k - 1)(n - k) \geq 0,$$

što je istina upravo za $k \in \{1, \dots, n\}$.

Mnogi problemi koje rješavamo računalom imaju složenost koja ne ovisi eksponencijalno o veličini problema n , nego polinomno, tj. broj aritmetičkih operacija proporcionalan je n^α , gdje je α neka mala konstanta (uobičajeno je $\alpha \leq 3$).

Vrlo je zanimljiv i problem prognoze vremena. On nam na suptilan način pokazuje što efikasnost znači u tom slučaju.

Primjer 1.1.2. (Prognoza vremena) *Najjednostavniji klimatski model funkcija je 4 argumenta: zemljopisne širine, dužine, visine (od tla) i vremena. Kao rezultat dobivamo 6 vrijednosti: temperaturu, tlak, vlažnost i brzinu vjetra (komponente u 3 smjera). Generalniji model mogao bi uključivati, na primjer, koncentraciju različitih plinova u atmosferi. Stvarni model atmosferskih procesa uključivao bi i stvaranje oblaka, količinu padalina, kemiju i sl.*

Primijetimo da se klima neprekidno mijenja (u ovisnosti o u svoje četiri varijable), ali je računalom možemo simulirati samo u ponekim (diskretnim) točkama.

Pretpostavimo da smo površinu Zemlje podijelili u (približne) kvadrate stranice 1 km. Po visini, također, uzimamo slojeve debljine 1 km, do visine 10 km. Stanje klime računamo samo u vrhovima kvadrata i točkama na slojevima iznad vrhova. Iz površine Zemlje ($\approx 5.1 \cdot 10^8 \text{ km}^2$) slijedi da je ukupan broj takvih točaka približno jednak $5 \cdot 10^9$.

Nadalje, klimu simuliramo u vremenskim trenucima s razmakom 1 minute. Za svaki vremenski trenutak i svaku prostornu točku moramo pamtit 6 vrijednosti koje opisuju stanje klime. Uzmimo da je svaka od njih realan broj koji se prikazuje s 4 byte-a. Onda je za pamćenje svih vrijednosti u jednom trenutku potrebno

$$6 \cdot 4 \cdot 5 \cdot 10^9 \approx 10^{11} \text{ B} = 0.1 \text{ TB}$$

memorije.

Simulacija klime napreduje po vremenu, tj. klima u sljedećem trenutku se računa iz stanja klime u nekoliko prethodnih trenutaka. Standardno se stanje klime u određenoj prostornoj točki računa iz stanja u nekoliko susjednih točaka. Pretpostavimo da nam za taj proračun treba samo 100 osnovnih aritmetičkih operacija (flopova) po svakoj točki, što ukupno daje $100 \cdot 5 \cdot 10^9 = 5 \cdot 10^{11}$ flopova.

Jasno je da za predviđanje vremena za sljedeću minutu ne smijemo potrošiti više od 1 minute vremena rada računala (inače je brže i jeftinije pogledati kroz prozor). Dakle, računalo mora imati brzinu od najmanje

$$5 \cdot 10^{11} \text{ flopova} / 60 \text{ sekundi} \approx 8 \cdot 10^9 \text{ flopsa} = 8 \text{ Gflopsa},$$

tj. preko 8 milijardi aritmetičkih operacija u sekundi.

Ako želimo dobiti globalnu prognozu vremena za samo 1 dan unaprijed, uz dozvoljeno vrijeme računanja od 2 sata (tako da ostane još 22 sata vrijednosti te prognoze), računalo mora biti još barem 12 puta brže, tj. brzina mora biti barem 100 Gflopsa ili 0.1 Tflopsa.

2. Greške

2.1. Računalo je izbacilo . . . Neće mi prihvatiti!

Koliko ste puta ove dvije rečenice čuli na TV-u, u banci ili negdje drugdje? U oba slučaja, računalo je personificirano kao nadnaravno sposobna osoba, koja je u prvom slučaju bezgrešna, a u drugom odbija nešto učiniti!

Treba li takvim tužbalicama vjerovati? I tko je krivac? Računalo ili ljudi koji su mu naredili da se upravo tako ponaša? Istina je da smo u 2001. godini, ali vaše računalo nije (svoje glavi) HAL 9000 iz “2001. odiseje u svemiru” (a i njemu su ljudi pomogli da postane ubojica).

Navedene dvije rečenice najčešće su samo jadno pokriće nečije nesposobnosti. Službeniku za šalterom u banci vjerojatno treba oprostiti, jer on samo izražava svoju bespomoćnost, a pravi krivac je negdje daleko.

Puno je opasnije kad čujete “Računalo je izbacilo . . .” od strane inženjera i znanstvenika kao glavno opravdanje budućih važnih projekata. Tad nas hvata strah! Zašto? Sama rečenica pokazuje da dobivene rezultate nitko nije pogledao, nego da im se slijepo vjeruje. A oni mogu biti pogrešni iz razno-raznih razloga, a najčešći krivac nije računalo.

Iz osobnog iskustva znamo da je provjera dobivenih rezultata najbolnija točka nastave numerike, u koju je slušače najteže uvjeriti. Metode je manje-više lako naučiti. U praksi, brojevi uvijek imaju neko značenje, izvora grešaka je puno — javljaju se na svakom koraku, a analiza grešaka u rezultatima katkad je vrlo sofisticirana. Slijepo vjerovanje rezultatima može biti pogibeljno.

Izvori grešaka 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.

No, ipak treba priznati da i računala, iznimno rijetko, ali ipak griješe. Koliko je nama poznato, u novija vremena poznata je samo greška dijeljenja u jednoj seriji Pentium procesora (1994. godine).

2.2. Mjere za grešku

Prije detaljnijeg opisa pojedinih vrsta ili uzroka grešaka, moramo preciznije reći što je greška i kako se ona uobičajeno mjeri.

Neka je x neki realni broj, kojeg smatramo “točnim”. Ako je \hat{x} neka njegova aproksimacija ili “približna vrijednost”, onda grešku te aproksimacije definiramo kao

$$\text{greška} = E(x, \hat{x}) := x - \hat{x},$$

tako da je $x = \hat{x} + \text{greška}$. Ovako definirana greška ima predznak i može biti negativna. U praksi nam, obično, predznak nije bitan kad govorimo o *točnosti* ove aproksimacije. Najkorisnije mjere za točnost ili grešku aproksimacije \hat{x} za x su:

- **apsolutna greška**

$$E_{\text{abs}}(x, \hat{x}) := |x - \hat{x}|,$$

koja mjeri stvarnu udaljenost (u smislu metrike na \mathbb{R}) brojeva x i \hat{x} , i

- **relativna greška**, definirana za $x \neq 0$,

$$E_{\text{rel}}(x, \hat{x}) := \frac{|x - \hat{x}|}{|x|},$$

koja mjeri relativnu točnost obzirom na veličinu broja x , na primjer, u smislu podudaranja “prednjih dijelova”, tj. određenog broja vodećih znamenki brojeva x i \hat{x} .

Ako aproksimaciju prikažemo u obliku $\hat{x} = x(1 + \rho)$, onda dobivamo ekvivalentnu definiciju relativne greške u obliku

$$E_{\text{rel}}(x, \hat{x}) := |\rho|.$$

Baš ovaj oblik se često koristi u analizi grešaka aritmetike računala. Ako želimo da relativna greška ima predznak, možemo ispustiti apsolutne vrijednosti iz definicije.

Na isti način možemo definirati i greške za kompleksne brojeve. Za teorijske potrebe to bi bilo dovoljno. Međutim, kao što ćemo vidjeti, kompleksne brojeve u računalu prikazujemo kao par realnih brojeva, tj. kao vektor s dvije realne komponente. Za mjerenje grešaka u vektorima i matricama, umjesto apsolutne vrijednosti, koristimo pojam norme, kojeg opisujemo u sljedećem poglavlju.

2.3. Greške modela

Najčešće greške modela nastaju zanemarivanjem utjecaja nekih sila (na primjer, zanemarivanje utjecaja otpora zraka). Jednako tako, da bi se dobilo nekakvo rješenje, barem približno, često se komplicirani model zamjenjuje jednostavnijim (na primjer, sistemi nelinearnih parcijalnih diferencijalnih jednačbi se lineariziraju).

Također, pogreške u modelu mogu nastati kod upotrebe modela u graničnim slučajevima. Na primjer, kod matematičkog njihala se $\sin x$ aproksimira s x , što vrijedi samo za male kuteve, a upotrebljava se, recimo, za kut od 15° .

Primjer 2.3.1. *Među prvim primjenama trenutno jednog od najbržih računala na svijetu bilo je određivanje trodimenzionalne strukture i elektronskog stanja ugljik-36 fulerena (engl. carbon-36 fullerene) — jednog od najmanjih, ali i najstabilnijih članova iz redova jedne vrste spojeva (engl. buckminsterfullerene). Primjena tog spoja može biti višestruka, od supravodljivosti na visokim temperaturama do preciznog doziranja lijekova u stanice raka.*

Prijašnja istraživanja kvantnih kemičara dala su dvije moguće strukture tog spoja. Eksperimentalna mjerenja pokazivala su da bi jedna struktura trebala biti stabilnija, a teoretičari su tvrdili da bi to trebala biti druga struktura. Naravno, te dvije strukture imaju različita kemijska svojstva. Prijašnja računanja, zbog pojednostavljivanja i interpolacije, kao odgovor davala su prednost “teoretskoj” strukturi. Definitivan odgovor, koji je proveden računanjem bez pojednostavljivanja pokazao je da je prva struktura stabilnija.

Svakako, treba istaknuti da su pogreške modela neuklonjive, ali zato je na inženjerima — korisnicima da procijene da li se primjenom danog modela dobivaju očekivani rezultati.

2.4. Greške u ulaznim podacima

Greške u ulaznim podacima javljaju se zbog nemogućnosti ili besmislenosti točnog mjerenja. Na primjer, tjelesna temperatura se obično mjeri na desetinku stupnja Celzusa točno — pacijentu je jednako loše ako ima tjelesnu temperaturu 39.5° ili 39.513462° .

Naravno, kao što ćemo to kasnije vidjeti, osim tih malih pogrešaka nastalih mjerenjem, dodatne greške nastaju spremanjem tih brojeva u računalo.

Vezano uz pogreške u ulaznim podacima, često se javlja pojam nestabilnog ili loše uvjetovanog problema. U praksi se vrlo često javljaju takvi problemi kod kojih mala perturbacija početnih podataka dovodi do velike promjene u rezultatu. Kao ilustraciju možemo uzeti sljedeći primjer.

Primjer 2.4.1. *Zadana su dva sistema linearnih jednadžbi*

$$\begin{aligned} 2x + 6y &= 8 \\ 2x + 6.0001y &= 8.0001, \end{aligned} \quad (2.4.1)$$

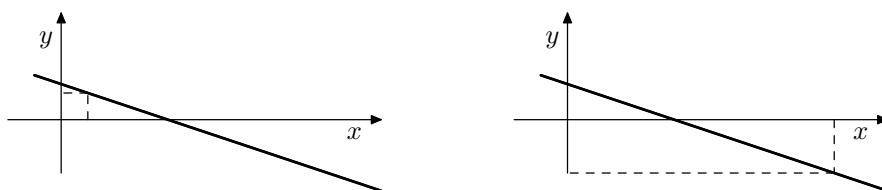
i

$$\begin{aligned} 2x + 6y &= 8 \\ 2x + 5.99999y &= 8.00002. \end{aligned} \quad (2.4.2)$$

Na prvi pogled, malom perturbacijom koeficijenata jednadžbe (2.4.1) dobivamo jednadžbu (2.4.2). Takva perturbacija mogla bi nastupiti, na primjer, malo pogrešno izmjerenim početnim podacima ili greškom koja je nastala u računu koeficijenata.

Što očekujemo? Očekujemo da će i rješenje drugog problema biti malo perturbirano rješenje prvog problema. Ali nije tako! Rješenje prvog problema je $x = 1$, $y = 1$, a drugog $x = 10$, $y = -2$! Zašto?

U analizi će nam pomoći crtanje odgovarajućih pravaca i njihovih sjecišta.



Ali na prethodnim slikama nacrtan je samo po jedan pravac! Pogrešno! Ako malo bolje pogledamo koeficijente u oba sistema jednadžbi, vidimo da se u oba slučaja radi o dva pravca koja su gotovo jednaka! Sad nije niti čudo da mala perturbacija koeficijenata pravca bitno udaljava presjecište.

2.5. Greške metoda za rješavanje problema

Greške metoda za rješavanje problema često nastaju kad se beskonačni procesi moraju zaustaviti u konačnosti. To vrijedi za sve objekte koji su definirani limesom — poput derivacija i integrala, i za sve postupke u kojima se “pravo” rješenje dobiva na limesu — konvergencijom niza približnih rješenja prema pravom. Velik broj numeričkih metoda za aproksimaciju funkcija i rješavanje jednadžbi upravo je tog oblika. Greške koja nastaju zamjenom beskonačnog nečim konačnim, obično dijelimo u dvije kategorije:

- **greške diskretizacije** (engl. discretization errors), koje nastaju zamjenom kontinuuma konačnim diskretnim skupom točaka, ili “beskonačno” malu veličinu h ili $\varepsilon \rightarrow 0$ zamijenjujemo nekim “konačno” malim brojem;

- **greške odbacivanja** (engl. truncation errors), koje nastaju “rezanjem” beskonačnog niza ili reda na konačni niz ili sumu, tj. odbacujemo ostatak niza ili reda.

Grubo rečeno, diskretizacija je vezana za kontinuum, a odbacivanje za diskretnu beskonačnost, poput razlike između skupova \mathbb{R} i \mathbb{N} .

Pojam diskretizacije smo već susreli u problemu prognoze vremena. Još jednostavniji, tipični primjer greške diskretizacije je aproksimacija funkcije f na intervalu (segmentu) $[a, b]$, vrijednostima te funkcije na konačnom skupu točaka (tzv. mreži) $\{x_1, \dots, x_n\} \subset [a, b]$, o čemu će još biti mnogo riječi.

Drugi klasični primjer je aproksimacija derivacije funkcije f u nekoj točki x . Po definiciji je

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h},$$

a za približnu vrijednost uzmemo neki dovoljno mali $h \neq 0$ i izračunamo kvocijent

$$f'(x) \approx \frac{\Delta f}{\Delta x} = \frac{f(x+h) - f(x)}{h}.$$

Postoje i bolje aproksimacije za $f'(x)$, ali o tome kasnije. Uskoro ćemo vidjeti da s ovom vrstom aproksimacija limesa treba vrlo oprezno postupati u aritmetici računala.

Pogledajmo malo i greške odbacivanja. Na primjer, beskonačna suma se mora zamijeniti konačnom, da bi se uopće mogla izračunati u konačnom vremenu.

Primjer 2.5.1. *Funkcije e^x i $\sin x$ imaju Taylorove redove oko točke 0 koji konvergiraju za proizvoljan $x \in \mathbb{R}$. Zbrajanjem dovoljno mnogo članova tih redova, možemo, barem u principu, dobro aproksimirati vrijednosti funkcija e^x i $\sin x$.*

Ako to napravimo računalom, rezultat će biti zanimljiv. Greška metode (tj. greška odbacivanja) lako se računa. Za dovoljno glatku funkciju f , Taylorov red

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} x^k$$

možemo aproksimirati Taylorovim polinomom p

$$f(x) = p(x) + R_{n+1}(x), \quad p(x) = \sum_{k=0}^n \frac{f^{(k)}(0)}{k!} x^k, \quad R_{n+1}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} x^{n+1},$$

pri čemu je ξ neki broj između 0 i x . Traženi Taylorovi polinomi s istim brojem članova (ali ne istog stupnja) su

$$e^x \approx \sum_{k=0}^n \frac{x^k}{k!}, \quad \sin x \approx \sum_{k=0}^n \frac{(-1)^k x^{2k+1}}{(2k+1)!}.$$

Nađimo grešku. Vrijedi

$$(e^x)^{(n)} = e^x, \quad (\sin x)^{(n)} = \sin\left(x + \frac{n\pi}{2}\right),$$

pa su pripadne greške odbacivanja

$$R_{n+1}(x) = \frac{e^\xi x^{n+1}}{(n+1)!}, \quad R_{2n+3}(x) = \frac{\sin\left(\xi + \frac{2n+3}{2}\pi\right)x^{2n+3}}{(2n+3)!},$$

Radi jednostavnosti, pretpostavimo da je $x > 0$. Iz $\xi \leq x$ dobivamo ocjene

$$|R_{n+1}(x)| \leq \frac{e^x x^{n+1}}{(n+1)!}, \quad |R_{2n+3}(x)| = \left| \frac{\sin\left(\xi + \frac{2n+3}{2}\pi\right)x^{2n+3}}{(2n+3)!} \right| \leq \left| \frac{x^{2n+3}}{(2n+3)!} \right|.$$

Zbrojimo li članove reda sve dok apsolutna vrijednost prvog odbačenog člana ne padne ispod zadane točnosti $\varepsilon > 0$, napravili smo grešku odbacivanja manju ili jednaku

$$\begin{cases} e^x \varepsilon, & \text{za } e^x, \\ \varepsilon, & \text{za } \sin x. \end{cases} \quad (2.5.1)$$

Izračunajmo $\sin(15\pi)$, $e^{15\pi}$, $\sin(25\pi)$ i $e^{25\pi}$ korištenjem Taylorovog reda oko 0 u najvećoj direktno podržanoj preciznosti (tzv. **extended** preciznosti). Primjer je izabran tako da je $\sin(15\pi) = \sin(25\pi) = 0$, dok su druga dva broja vrlo velika. Prema (2.5.1), greška metode za računanje je u slučaju funkcije e^x relativno mala, a u slučaju funkcije $\sin x$, mala po apsolutnoj vrijednosti.

Izaberemo li $\varepsilon = 10^{-17}$, dobivamo (napisano je samo prvih par znamenki rezultata)

$\sin(15\pi)_{\text{funkcija}} = 9.3241 \cdot 10^{-18}$	$\exp(15\pi)_{\text{funkcija}} = 2.9218 \cdot 10^{20}$
$\sin(15\pi)_{\text{Taylor}} = -2.8980 \cdot 10^{-1}$	$\exp(15\pi)_{\text{Taylor}} = 2.9218 \cdot 10^{20}$
$ \text{greška odbacivanja} \leq 2.7310 \cdot 10^{-19}$	$ \text{greška odbacivanja} \leq 2.7600 \cdot 10^2$
$\text{relativna greška} = 3.1081 \cdot 10^{16}$	$\text{relativna greška} = 1.4238 \cdot 10^{-18}$
$ \text{maksimalni član} = 1.6969 \cdot 10^{19}$	$ \text{maksimalni član} = 1.6969 \cdot 10^{19}$
$\sin(25\pi)_{\text{funkcija}} = 1.6697 \cdot 10^{-17}$	$\exp(25\pi)_{\text{funkcija}} = 1.2865 \cdot 10^{34}$
$\sin(25\pi)_{\text{Taylor}} = 3.0613 \cdot 10^{13}$	$\exp(25\pi)_{\text{Taylor}} = 1.2865 \cdot 10^{34}$
$ \text{greška odbacivanja} \leq 5.8309 \cdot 10^{-19}$	$ \text{greška odbacivanja} \leq 2.3782 \cdot 10^{16}$
$\text{relativna greška} = 1.8334 \cdot 10^{30}$	$\text{relativna greška} = 7.0013 \cdot 10^{-19}$
$ \text{maksimalni član} = 5.7605 \cdot 10^{32}$	$ \text{maksimalni član} = 5.7943 \cdot 10^{32}$

Ako smo rezultat zbrajanja Taylorovog reda za $\sin(15\pi)$ spremni prihvatiti kao približno točan, sigurno nije istina da je $\sin(25\pi) \approx 3 \cdot 10^{13}$. Što se, zapravo, dogodilo? Objasnjenje leži u aritmetici računala.

2.6. Greške aritmetike računala

U računalu postoje dva bitno različita tipa brojeva: cijeli brojevi i realni brojevi. Oba skupa su **konačni podskupovi** odgovarajućih skupova \mathbb{Z} i \mathbb{R} u matematici. Kao baza za prikaz oba tipa koristi se baza 2.

Cijeli se brojevi prikazuju korištenjem n bitova — binarnih znamenki, od kojih jedna služi za predznak, a ostalih $n - 1$ za znamenke broja. Matematički gledano, aritmetika cijelih brojeva u računalu je modularna aritmetika u prstenu ostataka modulo 2^n , samo je sistem ostataka simetričan oko 0, tj.

$$-2^{n-1}, \dots, -1, 0, 1, \dots, 2^{n-1} - 1.$$

Brojeve izvan tog raspona uopće ne možemo spremiti u računalu.

Realni brojevi r prikazuju se korištenjem mantise m i eksponenta e u obliku

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

pri čemu je e cijeli broj u određenom rasponu, a m racionalni broj za koji vrijedi $1/2 \leq m < 1$ (tj. mantisa započinje s 0.1...). Često se i ta vodeća jedinica ne pamti, jer se zna da su brojevi normalizirani, pa se mantisa “umjetno” može produžiti za taj jedan bit. Taj bit se katkad zove “skriveni bit” (engl. hidden bit). Za $r = 0$, mantisa je 0. U računalu se eksponent prikazuje kao s -bitni cijeli broj, a za mantisu pamti se prvih t znamenki iza binarne točke.

mantisa					eksponent				
±	m_{-1}	m_{-2}	⋯	m_{-t}	±	e_{s-2}	e_{s-3}	⋯	e_0

Dakle, skup svih realnih brojeva prikazivih u računalu je omeđen, a možemo ga parametrizirati duljinom mantise i eksponenta i označiti s $\mathbb{R}(t, s)$.

Primijetite da se ne može svaki realni broj egzaktno spremiti u računalu. Pretstavimo da je broj $x \in \mathbb{R}$ unutar prikazivog raspona i

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

Ako mantisa broja ima više od t znamenki, bit će spremljena aproksimacija tog broja $f\ell(x) \in \mathbb{R}(t, s)$ koja se može prikazati kao

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

Slično kao kod decimalne aritmetike (kad je prva odbačena znamenka ≤ 4 zaokružujemo nadalje, inače nagore), ako je prva odbačena znamenka 1, broj zaokružujemo

nagore, a ako je 0 nadolje. Time smo napravili apsolutnu grešku manju ili jednaku 2^{-t-1+e} . Gledajući relativno, greška je manja ili jednaka

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

tj. imamo vrlo malu relativnu grešku. Veličinu 2^{-t} zovemo jedinična greška zaokruživanja (engl. unit roundoff) i uobičajeno označavamo s u .

Dakle, 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

$$fl(x) = (1 + \varepsilon)x, \quad |\varepsilon| \leq u, \quad (2.6.1)$$

gdje je ε relativna greška napravljena tim zaokruživanjem.

Ovakav prikaz realnih brojeva u računalu zove se *prikaz s pomičnim zarezom* ili *točkom* (engl. floating point representation), a pripadna aritmetika je *aritmetika pomičnog zareza* ili *točke* (engl. floating point arithmetic).

Da bismo stekli osjećaj o veličinama o kojim govorimo, napišimo s i t i veličine koje se iz njih izvode za standardne tipove realnih brojeva u računalu:

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

Za sva tri tipa u ukupnoj duljini 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 pamtit. To je dogovoreni IEEE standard u kojem je propisano, ne samo kako se brojevi prikazuju u računalu, nego i svojstva aritmetike. Budimo pošteni, taj standard je nešto složeniji nego što smo to ovdje opisali. Međutim, ti dodatni detalji, poput posebnih prikaza za $\pm\infty$ i “rezultat nedozvoljene operacije” (engl. NaN, Not-a-Number), ili “zaštitne znamenke” (engl. guard digit) u aritmetici, nepotrebno zamagljuju bitno.

Osnovna pretpostavka ovog standarda je da za osnovne aritmetičke operacije (\circ označava $+$, $-$, $*$, $/$) nad $x, y \in \mathbb{R}(t, s)$ vrijedi

$$fl(x \circ y) = (1 + \varepsilon)(x \circ y), \quad |\varepsilon| \leq u, \quad (2.6.2)$$

za sve $x, y \in \mathbb{R}(t, s)$ za koje je $x \circ y$ u dozvoljenom rasponu. Naravno, dobiveni rezultat je tada prikaziv, tj. vrijedi $fl(x \circ y) \in \mathbb{R}(t, s)$. U protivnom, postoje rezervirani eksponenti koji označavaju “posebno stanje” (overflow, underflow, dijeljenje s 0 i nedozvoljenu operaciju kao što su $0/0$, $\sqrt{-1}$).

Oznaka $fl(\)$ sad ima značenje rezultata dobivenog računalom za operaciju $x \circ y$. Ovaj model kaže da je izračunata vrijednost za $x \circ y$ “jednako dobra” kao i zaokružen egzaktni rezultat, u smislu da je u oba slučaja jednaka ocjena relativne greške. Model, međutim, ne zahtijeva da za egzaktno prikazivi egzaktni rezultat $x \circ y \in \mathbb{R}(t, s)$ mora vrijediti da je greška $\varepsilon = 0$, tj.

$$x \circ y \in \mathbb{R}(t, s) \not\Rightarrow fl(x \circ y) = x \circ y.$$

U tom smislu, korištenje oznake $fl(x \circ y)$ za izračunati rezultat nije sasvim korektno, jer to ne mora biti zaokružen egzaktni rezultat. Preciznije bi bilo uvesti posebne oznake \oplus , \ominus , \otimes i \oslash za mašinske aritmetičke operacije i analizirati njihova svojstva. Takve analize postoje, ali su izuzetno komplicirane, jer većina standardnih svojstava aritmetičkih operacija, poput asocijativnosti zbrajanja ili distributivnosti množenja prema zbrajanju, **ne** vrijedi za aritmetiku realnih brojeva u računalu. Može se pokazati da vrijede neka bitno složenija “zamjenska” svojstva. Međutim, njih je nemoguće praktično iskoristiti za analizu ukupnih grešaka računanja u bilo kojem algoritmu s iole većim brojem aritmetičkih operacija.

Upravo zbog toga, standard za aritmetiku računala propisuje samo da mora vrijediti relacija (2.6.2), a ne neka druga složenija svojstva. Vidimo da greška svake pojedine aritmetičke operacije i njena ocjena u (2.6.2) imaju isti oblik kao i greška zaokruživanja za prikaz brojeva u računalu i njena ocjena iz (2.6.1). Zato obje vrste grešaka (greške prikaza i greške aritmetike) zajedničkim imenom zovemo greškama zaokruživanja (engl. rounding errors).

Objasnimo još točno značenje oznake $fl(\)$. Jednostavno, $fl(\text{izraz})$ označava **izračunatu** vrijednost tog izraza u floating point aritmetici. U skladu s tim, ako se izraz sastoji samo od jednog broja x , onda se “računanje” vrijednosti tog izraza x svodi na zaokruživanje i spremanje u floating point prikazu, a $fl(x)$ označava spremljenu “izračunatu”, tj. zaokruženu vrijednost. Analogno, $fl(x \circ y)$ sad korektno označava izračunati rezultat operacije $x \circ y$. Takva interpretacija znatno olakšava zapis u analizi grešaka zaokruživanja.

2.7. Propagiranje grešaka u aritmetičkim operacijama

Desnu stranu relacije (2.6.2) možemo interpretirati i kao egzaktno izvedenu operaciju \circ na malo perturbiranim podacima. Koje su operacije opasne ako nam je aritmetika egzaktna, a podaci malo perturbirani, tj. ako je $|\varepsilon_x|, |\varepsilon_y| \leq u$? Ako je \circ množenje, imamo

$$x(1 + \varepsilon_x) * y(1 + \varepsilon_y) \approx xy(1 + \varepsilon_x + \varepsilon_y) := xy(1 + \varepsilon_*), \quad |\varepsilon_*| \leq 2u.$$

Ako imamo dijeljenje, vrijedi slično

$$\frac{x(1 + \varepsilon_x)}{y(1 + \varepsilon_y)} \approx \frac{x}{y} (1 + \varepsilon_x)(1 - \varepsilon_y) := \frac{x}{y} (1 + \varepsilon_l), \quad |\varepsilon_l| \leq 2u.$$

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

$$x(1 + \varepsilon_x) + y(1 + \varepsilon_y) = x + y + x\varepsilon_x + y\varepsilon_y := (x + y) \left(1 + \frac{x\varepsilon_x + y\varepsilon_y}{x + y} \right),$$

uz pretpostavku da $x + y \neq 0$. Definiramo

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

Ako su x i y brojevi istog predznaka, onda je

$$\left| \frac{x}{x + y} \right|, \left| \frac{y}{x + y} \right| \leq 1, \quad (2.7.1)$$

pa je $|\varepsilon_{\pm}| \leq 2u$. U suprotnom, ako x i y imaju različite predznake, kvocijenti u (2.7.1) mogu biti proizvoljno veliki kad je $|x + y| \ll |x|, |y|$.

Možemo zaključiti da **opasnost** nastupa ako je rezultat zbrajanja brojeva suprotnog predznaka broj koji je po apsolutnoj vrijednosti mnogo manji od polaznih podataka. Dakle, čak i kad bi aritmetika računala bila egzaktna, zbog početnog zaokruživanja, rezultat može biti (i najčešće je) pogrešan.

Pokažimo to na jednostavnom primjeru računala u bazi 10. Pretpostavimo da je mantisa duga 4 dekadске znamenke, a eksponent dvije. Neka je

$$x = 0.88866 = 0.88866 \cdot 10^0, \quad y = 0.88844 = 0.88844 \cdot 10^0.$$

Umjesto brojeva x i y , spremili smo najbliže prikazive, tj.

$$fl(x) = 0.8887 \cdot 10^0, \quad fl(y) = 0.8884 \cdot 10^0$$

i napravili malu relativnu grešku. Budući da su im eksponenti jednaki, možemo oduzeti znamenku po znamenku mantise, a zatim normalizirati i dobiti

$$fl(x) - fl(y) = 0.8887 \cdot 10^0 - 0.8884 \cdot 10^0 = 0.0003 \cdot 10^0 = 0.3???? \cdot 10^{-3},$$

pri čemu upitnici predstavljaju znamenke koje više ne možemo restaurirati, pa računalo na ta mjesta upisuje 0. Primijetimo da je pravi rezultat $0.22 \cdot 10^{-3}$, pa je već prva značajna znamenka pogrešna, a relativna greška velika!

Iako je sama operacija oduzimanja bila egzaktna za $fl(x)$ i $fl(y)$, 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 2.7.1. *Objašnjenje pogrešnih rezultata iz primjera 2.5.1. sad je sasvim jednostavno. Pogledamo li po apsolutnoj vrijednosti najveće brojeve koji se javljaju u računu, ustanovljavamo da su oni golemi. Za $\sin x$, rezultat je malen broj koji je dobiven oduzimanjem velikih brojeva, pa je netočan. Nasuprot tome, kod funkcije e^x , uvijek imamo zbrajanje brojeva istog predznaka, pa je rezultat točan.*

Primjer 2.7.2. *U algoritmima se često javlja potreba za izračunavanjem vrijedosti $y = \sqrt{x + \delta} - \sqrt{x}$ uz $|\delta| \ll x$, $x > 0$. Da bismo izbjegli katastrofalno kraćenje, y se nikad ne računa po napisanoj formuli. Uvijek se pribjegava deracionalizaciji, tj.*

$$y = (\sqrt{x + \delta} - \sqrt{x}) \cdot \frac{\sqrt{x + \delta} + \sqrt{x}}{\sqrt{x + \delta} + \sqrt{x}} = \frac{\delta}{\sqrt{x + \delta} + \sqrt{x}}.$$

Uočite da je opasno oduzimanje zamijenjeno benignim dijeljenjem i zbrajanjem.

Primjer 2.7.3. *Zbrajanje brojeva računalom nije asocijativno. Izračunajmo*

$$S_1 = \sum_{i=1}^{10000} \frac{1}{i}, \quad S_2 = \sum_{i=10000}^1 \frac{1}{i}$$

u tri točnosti. Dobiveni rezultati su:

	single	double	extended
S_1	9.78761291503906250	9.78760603604434465	9.78760603604438230
S_2	9.78760433197021484	9.78760603604438550	9.78760603604438227

Primijetite da nešto točniji rezultat daje zbrajanje S_2 . Objasnite.

Primjer 2.7.4. *Niti poredak operacija ne mora biti beznačajan. Zadan je linearni sustav*

$$\begin{aligned} 0.0001 x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2. \end{aligned}$$

Matrica sustava je regularna, pa postoji jedinstveno rješenje $x_1 = 1.0001$, $x_2 = 0.9999$. Rješavamo li taj sustav računalom koje ima 4 decimalne znamenke mantise i 2 znamenke eksponenta, onda njegovo rješenje ovisi o poretku jednadžbi. Sustav zapisan u takvom računalu pamti se kao

$$\begin{aligned} 0.1000 \cdot 10^{-3} x_1 + 0.1000 \cdot 10^1 x_2 &= 0.1000 \cdot 10^1 \\ 0.1000 \cdot 10^1 x_1 + 0.1000 \cdot 10^1 x_2 &= 0.2000 \cdot 10^1. \end{aligned} \quad (2.7.2)$$

Prvo, riješimo sustav (2.7.2) Gausovim eliminacijama. Množenjem prve jednadžbe s 10^4 i oduzimanjem od druge, dobivamo drugu jednadžbu oblika:

$$(0.1000 \cdot 10^1 - 0.1000 \cdot 10^5) x_2 = 0.2000 \cdot 10^1 - 0.1000 \cdot 10^5. \quad (2.7.3)$$

Da bi računalo moglo oduzeti odgovarajuće brojeve, manji eksponent mora postati jednak većem, a mantisa se denormalizira. Dobivamo

$$0.1000 \cdot 10^1 = 0.0100 \cdot 10^2 = 0.0010 \cdot 10^3 = 0.0001 \cdot 10^4 = 0.0000|1 \cdot 10^5,$$

ali za zadnju jedinicu nema mjesta u mantisi, pa je mantisa postala 0. Slično je i s desnom stranom. Zbog toga jednačba (2.7.3) postaje

$$-0.1000 \cdot 10^5 x_2 = -0.1000 \cdot 10^5,$$

pa joj je rješenje $x_2 = 0.1000 \cdot 10^1$. Uvrštavanjem u prvu jednačbu, dobivamo:

$$0.1000 \cdot 10^{-3} x_1 = -0.1000 \cdot 10^1 \cdot 0.1000 \cdot 10^1 + 0.1000 \cdot 10^1 = 0.0000,$$

pa je $x_1 = 0$, što nije niti približno točan rezultat.

Promijenimo li poredak jednačbi u (2.7.2), dobivamo

$$\begin{aligned} 0.1000 \cdot 10^1 x_1 + 0.1000 \cdot 10^1 x_2 &= 0.2000 \cdot 10^1 \\ 0.1000 \cdot 10^{-3} x_1 + 0.1000 \cdot 10^1 x_2 &= 0.1000 \cdot 10^1. \end{aligned} \quad (2.7.4)$$

Množenjem prve jednačbe s 10^{-4} i oduzimanjem od druge, dobivamo drugu jednačbu oblika

$$(0.1000 \cdot 10^1 - 0.1000 \cdot 10^{-3}) x_2 = 0.1000 \cdot 10^1 - 0.2000 \cdot 10^{-3}, \quad (2.7.5)$$

pa se (2.7.5) svede na $0.1000 \cdot 10^1 x_2 = 0.1000 \cdot 10^1$, tj. $x_2 = 0.1000 \cdot 10^1$. Uvrštavanjem u prvu jednačbu u (2.7.4) dobivamo

$$0.1000 \cdot 10^1 x_1 = 0.2000 \cdot 10^1 - 0.1000 \cdot 10^1 \cdot 0.1000 \cdot 10^1 = 0.1000 \cdot 10^1,$$

pa je $x_1 = 0.1000 \cdot 10^1$, što točan rezultat korektno zaokružen na četiri decimalne znamenke.

Primjer 2.7.5. Poznato je da studenti kad ne znaju izračunati limese, posežu za kalkulatorom i pokušavaju ih “heuristički” izračunati, tako da se približavaju granici. Pretpostavimo da imaju program koji u **extended** točnosti računa sljedeće limese:

$$\lim_{x \rightarrow 0} \frac{1 - \cos x}{x^2} = \frac{1}{2}, \quad \lim_{x \rightarrow 0} \frac{x^2}{1 - \cos x} = 2.$$

Može li se unaprijed reći što će se događati s rezultatima ako stavljamo x -eve redom

$10^{-1}, 10^{-2}, \dots, 10^{-10}$? Objasnite dobivene rezultate.

x	prvi ‘limes’	drugi ‘limes’
10^{-01}	0.4995834722	2.001667500
10^{-02}	0.4999958333	2.000016667
10^{-03}	0.4999999583	2.000000167
10^{-04}	0.4999999996	2.000000002
10^{-05}	0.5000000002	1.999999999
10^{-06}	0.4999999980	2.000000008
10^{-07}	0.5000015159	1.999993936
10^{-08}	0.4998172015	2.000731461
10^{-09}	0.4336808690	2.305843009
10^{-10}	0.0000000000	

2.8. Primjeri iz života

Primjeri koje smo naveli u prethodnom odjeljku su pravi, školski. Nažalost, postoje primjeri kad su zbog grešaka zaokruživanja stradali ljudi ili je počinjena velika materijalna šteta.

Primjer 2.8.1. (Promašaj raketa Patriot.)

U Zaljevskom ratu, 25. veljače 1991. godine, Patriot rakete iznad Dhahrana u Saudijskoj Arabiji nisu uspjele pronaći i oboriti iračku Scud raketu. Projektil je (pukim slučajem) pao na američku vojnu bazu usmrтивši 28 i ranivši stotinjak ljudi.

Izvještaj o katastrofi godinu dana poslije, rasvijetlio je okolnosti nesreće. U računalu koje je upravljalo Patriot raketama, vrijeme se brojilo u desetinkama sekunde proteklim od trenutka kad je računalo upaljeno. Kad desetinku prikazemo u binarnom prikazu, dobivamo

$$0.1_{10} = (0.00011)_2.$$

Realne brojeve u tom računalu prikazivali su korištenjem nenormalizirane mantise duljine 23 bita. Spremanjem 0.1 u registar Patriot računala, napravljena je (apsolutna) greška približno jednaka $9.5 \cdot 10^{-8}$.

Zbog stalne opasnosti od napada Scud raketama, računalo je bilo u pogonu 100 sati, što je $100 \cdot 60 \cdot 60 \cdot 10$ desetinki sekunde. Ukupna greška nastala greškom zaokruživanja je

$$100 \cdot 60 \cdot 60 \cdot 10 \cdot 9.5 \cdot 10^{-8} = 0.34 \text{ s.}$$

Ako je poznato da Scud putuje brzinom 1676 m/s na predviđenoj visini susreta, onda su ga rakete Patriot pokušale naći više od pola kilometra daleko od njegovog stvarnog položaja.

Koje je precizno objašnjenje uzroka ove katastrofe? Stvarni uzrok je nedovoljno pažljivo, “mudro” ili “hackersko” pisanje programa. Patriot sustav prati cilj tako da mjeri vrijeme potrebno radarskim signalima da se odbiju od cilja i vrate natrag. Točnost podataka o vremenu je, naravno, ključna za precizno uništenje cilja.

Računalo koje je upravljalo Patriot raketama bazirano je na konstrukciji iz 1970-ih godina i koristi 24-bitnu aritmetiku. Međutim, realizacija floating point aritmetike u ta “davna” vremena bila je mnogo sporija od cjelobrojne, posebno za množenje i dijeljenje. Zbog toga se u programima često koristila kombinacija cjelobrojne (tzv. fixed point) aritmetike i floating point aritmetike, da se ubrzaju te dvije operacije.

Tako je sistemski sat mjerio vrijeme u desetinkama sekunde, ali se vrijeme spremalo kao cijeli broj desetinki proteklih od trenutka kad je računalo upaljeno. Za sve ostale proračune, vrijeme se računa tako da se pomnoži broj desetinki n s osnovnom jedinicom $t_0 = 0.1$ s u kvazi-cjelobrojnoj aritmetici, a zatim pretvori u pravi 24-bitni floating point broj.

Kvazi-cjelobrojni ili fixed point prikaz realnog broja odgovara prikazu cijelih brojeva, s tim da se uzima da je binarna točka ispred prve prikazane znamenke. Preciznije rečeno, ako imamo 24 bita za takav prikaz, realni broj se interpretira kao višekratnik od 2^{-23} , a prikaz se dobiva zaokruživanjem višekratnika na cijeli broj i to odbacivanjem racionalnog dijela (u smjeru nule). Dakle, za realni broj $r \geq 0$ pamti se cijeli broj $\lfloor r \cdot 2^{23} \rfloor$, a za negativni r pamti se $-\lfloor |r| \cdot 2^{23} \rfloor$. Naravno, tako se mogu prikazati samo realni brojevi iz $[-1, 1)$, inače imamo premalo bitova.

Za analizu grešaka, ignorirajmo da se stvarno sprema cijeli broj, i pogledajmo kojoj aproksimaciji za r odgovara taj spremljeni broj. Neka $fi(r)$ označava tu aproksimaciju za r , u smislu da je spremljeni prikaz od r , zapravo, egzaktni prikaz broja $fi(r)$. Taj broj možemo jednostavno dobiti tako da spremljene bitove interpretiramo kao prikaz s nenormaliziranom mantisom od 23 bita, jer se pamte redom znamenke iza binarne točke, a eksponenta nema, tj. jednak je 0. Dakle, za $r \in [-1, 1)$, broj $fi(r)$ ima točno prva 23 bita od r iza binarne točke, a ostatak zanemarujemo, zbog zaokruživanja odbacivanjem. Za apsolutnu grešku, očito, vrijedi

$$|x - fi(x)| < 2^{-23},$$

ali relativna greška može biti velika.

Kako to izgleda za osnovnu vremensku jedinicu $t_0 = 0.1$? Zadržavanjem prva 23 bita iza binarne točke i odbacivanjem ostatka u

$$0.1 = (0.0001\ 1001\ 1001\ 1001\ 1001\ 1001\ 100|1\ 1001\ \dots)_2.$$

dobivamo

$$fi(0.1) = (0.0001\ 1001\ 1001\ 1001\ 1001\ 100)_2.$$

Učinjena (apsolutna) greška je

$$|0.1 - fi(0.1)| = 0.1 \cdot 2^{-20} = \frac{1}{10485760} = 9.5367431640625 \cdot 10^{-8},$$

dok je relativna greška točno 2^{-20} , ili 10 puta veća, što uopće ne izgleda strašno.

Nakon n otkucaja sistemskog sata (u desetinkama), pravo vrijeme u sekundama je $t = n \cdot t_0$. Umjesto toga, računa se $n \cdot fi(t_0)$. Pretpostavimo da se to množenje izvodi egzaktno, bez dodatnih grešaka zaokruživanja (kao da smo u cjelobrojnoj aritmetici). Izračunato vrijeme je $\hat{t} = n \cdot fi(t_0)$. Relativna greška ostaje ista

$$\left| \frac{t - \hat{t}}{t} \right| = 2^{-20},$$

jer se n skrati. Međutim, apsolutna greška je n puta veća

$$|t - \hat{t}| = n \cdot |0.1 - fi(0.1)| \approx n \cdot (9.5 \cdot 10^{-8}).$$

Nažalost, za točno gađanje treba apsolutna, a ne relativna točnost u vremenu. Za dovoljno veliki n , a nakon 100 sati je $n = 3600000$, dobivenom \hat{t} nema spasa, čak i prije pretvaranja u floating point prikaz (što još doprinosi ukupnoj pogrešci).

Što se moglo napraviti? Da su se ista 23 bita koristila za pravu mantisu u floating point prikazu

$$0.1 = (0.1100\ 1100\ 1100\ 1100\ 1100\ 110|0\ 1100\ \dots)_2 \cdot 2^{-3},$$

bez obzira na vrstu zaokruživanja, dobili bismo

$$fl(0.1) = (0.1100\ 1100\ 1100\ 1100\ 1100\ 110)_2 \cdot 2^{-3},$$

uz točno $2^4 = 16$ puta manje greške. Na primjer, apsolutna greška je

$$|0.1 - fl(0.1)| = 0.1 \cdot 2^{-24} \approx 5.96 \cdot 10^{-9}.$$

Čak i nakon 100 sati, posljedica ove greške je promašaj od oko 40m, što je (s visokom vjerojatnošću) još uvijek dovoljno točno za uništenje Scuda.

S druge strane, treba izbjegavati eksplicitno korištenje tzv. apsolutnog vremena od trenutka uključenja. Puno bolje je brojač iznova postaviti na nulu u trenutku prvog radarskog kontakta, ili zapamtiti stanje cjelobrojnog brojača u tom trenutku, a sva ostala vremena računati prvo cjelobrojnim oduzimanjem stanja brojača, pa tek onda dobivenu razliku pretvoriti u sekunde (pomnožiti bitno manji broj s t_0). Scud ipak leti malo kraće od 100 sati!

Zanimljivo je da je prva indikacija ove pogreške prijavljena punih 14 dana ranije, 11. veljače. Na jednom sustavu Patriota uočen je pomak u tzv. “prostoru udara” (engl. range gate) za punih 20% nakon neprekidnog rada od 8 sati. Ti podaci pokazivali su da nakon 20 sati neprekidnog rada, sustav neće moći pratiti i presteti nadolazeći Scud. Modificirani program koji kompenzira netočno računanje vremena službeno je izašao 16. veljače. Međutim, u Dhahran je stigao tek 26. veljače, dan nakon nesreće. Ipak, čudno je da posade sustava na terenu nisu dobile barem obavijest o problemu — povremeni “restart” sustava bio bi sasvim dovoljan za prvo vrijeme.

Primjer 2.8.2. (Eksplozija Ariane 5.)

Raketa Ariane 5 lansirana 4. lipnja 1995. godine iz Kouroua (Francuska Gvajana) nosila je u putanju oko Zemlje komunikacijske satelite vrijedne oko 500 milijuna USD. Samo 37 sekundi nakon lansiranja izvršila je samouništenje.

Dva tjedna kasnije, stručnjaci su objasnili događaj. Kontrolna varijabla (koja je služila samo informacije radi) u programu vođenja rakete mjerila je horizontalnu brzinu rakete. Greška je nastupila kad je program pokušao pretvoriti preveliki 64-bitni realni broj u 16-bitni cijeli broj. Računalo je javilo grešku, što je izazvalo samouništenje. Zanimljivo je da je taj isti program bio korišten u prijašnjoj sporijoj verziji Ariane 4, pa do katastrofe nije došlo.

Primjer 2.8.3. (Potonuće naftne platforme Sleipner A.)

Naftna platforma Sleipner A potonula je prilikom sidrenja, 23. kolovoza 1991. u blizini Stavangera. Baza platforme su 24 betonske ćelije, od kojih su 4 produljene u šuplje stupove na kojima leži paluba. Prilikom uronjavanja baze došlo je do pucanja. Rušenje je izazvalo potres jačine 3.0 stupnja po Richterovoj ljestvici i štetu od 700 milijuna USD.

Greška je nastala u projektiranju, primjenom standardnog paketa programa, kad je upotrijebljena metoda konačnih elemenata s nedovoljnom točnošću (netko nije provjerio rezultate programa). Proračun je dao naprezanja 47% manja od stvarnih. Nakon detaljne analize s točnijim konačnim elementima, izračunato je da su ćelije morale popustiti na dubini od 62 metra. Stvarna dubina pucanja bila je 65 metara!

Primjer 2.8.4. (Izabran je pogrešan predsjednik.)

Možda je najbizarniji primjer da greška zaokruživanja može poremetiti izbore za predsjednika SAD. U američkom sustavu izbora predsjednika, svaka od saveznih država ima određen broj predstavnika (ljudi) u tijelu koje se zove Electoral College i koje formalno bira predsjednika. Broj predstavnika svake pojedine države u tom tijelu proporcionalan je broju stanovnika te države u odnosu na ukupan broj stanovnika. Pretpostavimo da u Electoral College-u ima a predstavnika, populacija SAD je p stanovnika, a država i ima p_i stanovnika. Broj predstavnika države i u Electoral

College-u trebao bi biti

$$a_i = \frac{p_i}{p} \cdot a.$$

Ali, predstavnici su ljudi, pa bi a_i morao biti cijeli broj. Zbog toga se a_i mora zaokružiti na cijeli broj \hat{a}_i po nekom pravilu. Naravno, na kraju mora biti $\sum_i \hat{a}_i = a$. Razumno i prirodno pravilo je:

- \hat{a}_i mora biti jedan od dva cijela broja koji su najbliži a_i (tzv. “uvjet kvote”).

Naime, pravilno zaokruživanje (kao kod prikaza brojeva) je možda najpravednije, ali ne mora dati $\sum_i \hat{a}_i = a$, pa se mora upotrijebiti slabije pravilo.

Međutim, broj stanovnika p_i se vremenom mijenja (a time i p). Isto tako, ukupni broj predstavnika a u tijelu se može promijeniti od jednih do drugih izbora. Zbog toga se dodaju još dva prirodna “politička” pravila:

- Ako se poveća ukupan broj predstavnika a , a svi ostali podaci se ne promijene, \hat{a}_i ne smije opasti (tzv. “monotonost predstavničkog tijela”).
- Ako je broj stanovnika države p_i porastao, a ostali podaci su nepromijenjeni, \hat{a}_i ne smije opasti (tzv. “monotonost populacije”).

Svrha je jasna, jer ljudi vole uspoređivati prošle i nove “kvote”. Nažalost, ne postoji metoda za određivanje broja predstavnika koja bi zadovoljavala sva tri kriterija.

U američkoj povijesti zaista postoji slučaj da je izabran “pogrešan” predsjednik. Samuel J. Tilden izgubio je izbore 1876. godine u korist Rutherforda B. Hayesa, samo zbog načina dodjele elektorskih glasova u toj prilici. Da stvar bude još zanimljivija, ta metoda dodjele glasova nije bila ona koju je propisivao zakon iz tog vremena.

Matematički gledano, problem izbornih sustava i raznih pravila za računanje broja predstavnika u predstavničkim tijelima, poput Sabora, nije trivijalan, a ozbiljno se proučava već stotinjak godina.

Uzmimo najjednostavniji “izborni sustav” u kojem vrijedi samo “uvjet kvote” i pretpostavimo da se on primjenjuje za svake izbore posebno (tj. vremenski lokalno), uz poznate podatke o broju stanovnika p_i u svim “izbornim jedinicama”. Kako biste što pravednije izračunali brojeve \hat{a}_i predstavnika svake jedinice?