SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODJEL

Vjeran Hari

Numerička analiza — Uvod, greške

knjiga

Zagreb, 2003.

Sadržaj

1. Uvodni dio

Numerička matematika je disciplina koja proučava i numerički rješava matematičke probleme koji se javljaju u znanosti, tehnici, gospodarstvu itd. Iako se ta disciplina najčešće povezuje sa numeričkim metodama, treba znati da bez dubljeg poznavanja samog problema koji rješavamo, nije moguće procijeniti je li neka metoda dobra u smislu da daje zadovoljavajuće točna rješenja u dovoljno kratkom vremenskom intervalu. O problemu koji se rješava treba znati barem neka svojstva: postoji li barem jedno rješenje, ako da, je li rješenje jedinstveno i što se pokazuje kao vrlo važno, kako se rješenje (ili rješenja) ponašaju kad se polazni podaci malo promijene (teorija perturbacije). Kad se konstruira neka metoda za dani problem otvaraju se pitanja konvergencije (da li niz aproksimacija teži prema rješenju), brzine konvergencije (linearna, kvadratična, kubična,...), adaptibilnot metode za specijalna računala (paralelna, vektorska, standardna), kompleksnost metode (broj računskih operacija, zauzeće memorije, dohvat operanada, prenos podataka po memoriji,...) te točnost odnosno stabilnost metode (koliko točnih značajnih znamenaka ima izračunato rješenje i o čemu to ovisi). Uz određivanje stabilnosti algoritma vezana je i analiza grešaka zaokruživanja koja pokazuje mogu li greške koje strojna aritmetika generira u toku računskog procesa bitno narušiti točnost izlaznih podataka.

U ovom dijelu knjige uvest ćemo oznake osnovnih pojmova koje ćemo koristiti u knjizi. Neke pojmove ćemo definirati, a za neke ćemo dati najvažnije tvrdnje odnosno rezultate.

1.1. Pomoćni rezultati iz analize

U ovom dijelu ćemo navesti bez dokaza nekoliko važnih teorema iz analize. Evo prvo oznaka važnijih skupova.

Sa $\mathbf{N}$ označavamo skup prirodnih brojeva, dakle $\{1, 2, \ldots\}$. Sa $\mathbf{N}_0$ označavamo skup nenegativnih cijelih brojeva, $\mathbf{N}_0 = \mathbf{N} \cup \{0\}$. Sa $\mathbf{Z}$ označavamo skup svih cijelih brojeva, dakle i pozitivnih, negativnih i nule. Sa $\mathbf{Q}$ označavamo skup racionalnih, a sa $\mathbf{Q}_0$ skup nenegativnih racionalnih brojeva. Sa $\mathbf{R}$ ($\mathbf{C}$) označavamo skup realnih (kompleksnih) brojeva.

Ako je $\mathcal{S} \subseteq \mathbf{C}$ (ili $\mathcal{S} \subseteq \mathbf{R}$), tada je zatvarač $\mathrm{Cl}\,\mathcal{S}$ skupa $\mathcal{S}$, skup svih gomilišta od $\mathcal{S}$. S obzirom da je svaka točka od $\mathcal{S}$ ujedno i gomilište od $\mathcal{S}$, vrijedi $\mathcal{S} \subseteq \mathrm{Cl}\,\mathcal{S}$. Zatvarač $\mathrm{Cl}\,\mathcal{S}$ je najmanji zatvoren skup koji sadrži $\mathcal{S}$. S druge strane, unutrašnjost skupa $\mathcal{S}$, $Int\,\mathcal{S}$ ili interior od $\mathcal{S}$ je najveći otvoren skup sadržan u $\mathcal{S}$. Jasno je da je $Int\,\mathcal{S} \subseteq \mathcal{S} \subseteq \mathrm{Cl}\,\mathcal{S}$ pri čemu jednakosti vrijede npr. za cijeli $\mathbf{C}$ (ili $\mathbf{R}$). Ako je recimo $\mathrm{Cl}\,\mathcal{S}$ zatvoreni krug, onda je $Int\,\mathcal{S}$ otvoreni krug (krug bez rubne kružnice), a $\mathcal{S}$ može biti npr. otvoreni krug sa bilo kojim točkama na rubu.

U numeričkoj matematici su posebno važni sljedeći teoremi o srednjim vrijednostima čiji dokazi se mogu pronaći u knjigama iz elementarne analize.

Teorem 1.1.1. (Međuvrijednost) *Neka je f realna neprekidna funkcija na konačnom segmentu $[a,b]$. Neka su*

$$m = \inf_{a \le x \le b} f(x) \quad i \quad M = \sup_{a \le x \le b} f(x)$$

infimum i supremum funkcije f. Tada za svaki realni broj β iz segmenta $[m,M]$, postoji barem jedan realni broj α iz $[a,b]$, takav da je

$$f(\alpha) = \beta\,.$$

Specijalno, postoje brojevi $\underline{x}$ i $\overline{x}$ iz $[a,b]$, takvi da je

$$m = f(\underline{x})\,, \quad M = f(\overline{x})\,.$$

Teorem 1.1.2. (Srednja vrijednost) *Neka je f realna funkcija neprekidna na konačnom segmentu $[a,b]$ i diferencijabilna na otvorenom intervalu (a,b). Tada postoji barem jedna točka $\xi \in (a,b)$, takva da je*

$$f(b) - f(a) = f'(\xi)(b-a)\,.$$

Teorem 1.1.3. (Integralna srednja vrijednost) *Neka je w nenegativna i integrabilna funkcija na segmentu $[a,b]$. Neka je f neprekidna na $[a,b]$. Tada postoji točka $\xi \in [a,b]$ takva da je*

$$\int_a^b f(x)w(x)dx = f(\xi)\int_a^b w(x)dx\,.$$

Jedan od najvažnijih alata u numeričkoj matematici je Taylorov teorem jer daje jednostavnu metodu aproksimacije funkcije f pomoću polinoma. Kako se vrijednost polinoma lako odredi pomoću Hornerovog algoritma, dobivamo način računavanja vrijednosti funkcije f u danoj točki x. Da bi mogli koristiti taj pristup aproksimacije funkcije, ona mora biti dovoljno glatka.

Teorem 1.1.4. (Taylorov teorem) *Neka f ima neprekidne derivacije do reda $n+1>0$ na segmentu $[a,b]$. Ako su $x, x_0 \in [a,b]$, tada je*

$$f(x) = p_n(x) + R_{n+1}(x) \tag{1.1.1}$$

$$p_n(x) = f(x_0) + \frac{(x-x_0)}{1!}f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \cdots + \frac{(x-x_0)^n}{2!}f^{(n)}(x_0) \tag{1.1.2}$$

$$R_{n+1}(x) = \frac{1}{n!}\int_{x>0}^{x}(x-t)^n f^{(n+1)}(t)dt \tag{1.1.3}$$

$$= \frac{(x-x_0)^{n+1}}{(n+1)!}f^{(n+1)}(\xi) \tag{1.1.4}$$

za neki ξ između x_0 i x.

Dokaz. Izvod relacija (1.1.1)–(1.1.3) koristi n puta parcijalno integriranje, polazeći od identiteta

$$f(x) = f(x_0) + \int_{x_0}^{x} f'(t)dt\,.$$

Drugi oblik ostatka $R_{n+1}(x)$ se dobije korištenjem teorema integralne srednje vrijednosti sa funkcijom $w(t) = (x-t)^n$. Puni dokaz se može naći u većini knjiga iz elementarne analize. ■

Polinom p_n se zove Taylorov razvoj funkcije f u točki x_0. Ako je f beskonačno derivabilna, polinom p_n prelazi u red potencija oblika $(x-x_0)^n$ koji se zove Taylorov red. Uz pomoć Taylorovog reda lako se dobiju poznate formule:

$$e^x = 1 + x + \frac{x^2}{2} + \cdots + \frac{x^n}{n!} + \frac{x^{n+1}}{(n+1)!}e^{\xi_x} \tag{1.1.5}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots + (-1)^n\frac{x^{2n}}{(2n)!} + (-1)^{n+1}\frac{x^{2n+2}}{(2n+2)!}\cos(\xi_x) \tag{1.1.6}$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{45!} - \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + (-1)^n\frac{x^{2n+1}}{(2n+1)!}\cos(\xi_x) \tag{1.1.7}$$

$$(1+x)^\alpha = 1 + \binom{\alpha}{1}x + \binom{\alpha}{2}x^2 + \cdots + \binom{\alpha}{n}x^n + \binom{\alpha}{n+1}\frac{x^{n+1}}{(1+\xi_x)^{n+1-\alpha}} \tag{1.1.8}$$

pri čemu je

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}, \quad k = 1, 2, 3, \ldots$$

za proizvoljni realni broj α. U svim slučajevima je nepoznata točka ξ_x smještena između 0 i x.

Sjetimo se formule za odsječak geometrijskog reda

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x}. \tag{1.1.9}$$

Ako razlomak na desnoj strani napišemo kao razliku razlomaka, te dio sa potencijom x^{n+1} prebacimo na drugu stranu jednakosti, dobivamo

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n \ + \frac{x^{n+1}}{1-x}, \quad x \neq 1, \tag{1.1.10}$$

što je drugi, jednostavniji, zapis razvoja 1.1.8 uz $\alpha = -1$ i $-x$ umjesto x. Kad pustimo da n raste bez granica i ako je x iz itervala $(-1, 1)$, formula (1.1.10) prelazi u

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n \ + \cdots, , \quad |x| < 1. \tag{1.1.11}$$

Kad isto načinimo u formulama za e^x, $\cos(x)$ i $\sin(x)$, ne trebamo ograničiti x jer pripadni redovi konvergiraju apsolutno za svaki x.

Iako su formule (1.1.1)–(1.1.4) vrlo korisne za dobivanje razvoja funkcije f oko točke x_0, često puta računanje viših derivacija pretstavlja problem zbog kompleksnosti računanja. U takvim slučajevima, možemo se koristiti već dobivenim razvojima. Sljedeći primjeri pokazuju kako se to može načiniti.

Primjer 1.1.1. *Da bi dobili razvoj funkcije $f(x) = e^{-x^2}$ oko 0, možemo se koristiti razvojem (1.1.5) u koji umjesto x uvrstimo $-x^2$,*

$$e^{-x^2} = 1 - x^2 + \frac{x^4}{2!} - \cdots + (-1)^{n+1}\frac{x^{2n+2}}{(n+1)!}e^{\xi_x}$$

pri čemu je $-x^2 \leq \xi_x \leq 0$. ■

Primjer 1.1.2. *Da bi dobili razvoj funkcije $f(x) = \operatorname{arctg}(x)$ oko 0, možemo se koristiti razvojem (1.1.11) u koji umjesto x uvrstimo $-t^2$,*

$$\frac{1}{1+t^2} = 1 - t^2 + t^4 - \cdots + (-1)^n t^{2n} \ + (-1)^{n+1}\frac{t^{2n+2}}{1+t^2}.$$

Integrirajući po t na intervalu $[0, x]$, dobivamo

$$\operatorname{arctg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots + (-1)^n \frac{x^{2n+1}}{2n+1} + (-1)^{n+1}\int_0^x \frac{t^{2n+2}}{1+t^2}dt.$$

Primjena teorema o integralnoj srednjoj vrijednosti daje

$$\int_0^x \frac{t^{2n+2}}{1+t^2}dt = \frac{x^{2n+3}}{2n+3}\cdot\frac{1}{1+\xi_x^2}\,, \quad \xi_x \text{ između } 0 \text{ i } x$$

■

Primjer 1.1.3. *Neka je* $f(x) = \int_0^1 sin(xt)dt$. *Koristeći razvoj funkcije* sin *u Taylorov red oko nule (vidi relaciju (1.1.7)) dobivamo*

$$\begin{aligned} f(x) &= \int_0^1 \left[xt - \frac{x^3t^3}{3!} + \cdots + (-1)^{n-1}\frac{(xt)^{2n-1}}{(2n-1)!} + (-1)^n\frac{(xt)^{2n+1}}{(2n+1)!}\right]dt \\ &= \sum_{j=1}^{n}(-1)^{j-1}\frac{x^{2j-1}}{(2j)!} + (-1)^n\frac{x^{2n+1}}{(2n+1)!}\int_0^1 t^{2n+1}\cos(\xi_{xt})dt \end{aligned}$$

pri čemu je ξ_{xt} *između* 0 *i* xt. *Integral u ostatku se jednostavno ocijeni sa* $1/(2n+2)$, *ali može se dovesti i na jednostavniji oblik. Najme, može se pokazati da je* $t \mapsto \xi_{xt}$ *neprekidna funkcija, pa se može primijeniti integralni teorem srednje vrijednosti i dobiti*

$$\int_0^1 sin(xt)dt = \sum_{j=1}^{n}(-1)^{j-1}\frac{x^{2j-1}}{(2j)!} + (-1)^n\frac{x^{2n+1}}{(2n+2)!}\cos(\xi_x)\,, \quad \xi_x \text{ između } 0 \text{ i } x\,.$$

■

Primjer 1.1.4. *Neka su* x_0, x_1, x_2 *različiti realni brojevi i* f *realna funkcija definirana na intervalu koji sadrži te točke. Veličine*

$$f[x_0,x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0} \quad i \quad f[x_0,x_1,x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0} \tag{1.1.12}$$

se zovu podijeljene razlike *prvog i drugog reda od* f. *Ako je* f *jedamput odnosno dvaput diferencijabilna na odgovarajućem intervalu, može se pokazati da je*

$$f[x_0,x_1] = f'(\xi), \quad f[x_0,x_1,x_2] = \frac{1}{2}f''(\eta) \tag{1.1.13}$$

pri čemu je ξ *između* x_0 *i* x_1, *a* η *između minimuma i maksimuma skupa* $\{x_0, x_1, x_2\}$. *Prva tvrdnja slijedi lako iz teorema o srednjoj vrijednosti. Kasnije ćemo pokazati opća svojstva podijeljenih razlika proizvoljnog reda, a time i drugu tvrdnju. Također, odmah se vidi da je* $f[x_0,x_1] = f[x_1,x_0]$. *Pokušajte pokazati da je* $f[x_0,x_1,x_2] = f[x_i,x_j,x_k]$ *za bilo koju permutaciju* i,j,k *niza* $(1,2,3)$. ■

Neka je sada f realna funkcija dviju varijabli, $f : \mathbf{R}^2 \mapsto \mathbf{R}$. Koristeći kartezijev koordinatni sustav Oxy, možemo svakom paru realnih brojeva (x, y) pridružiti točno jednu točku u Kartezijevoj ravnini koju također označimo s (x, y) (apscisa te točke ima vrijednost x, a ordinatu ima vrijednost y). Kako vrijedi i obrat, par realnih brojeva $x, y)$ se poistovjećuje sa točkom ravnine.

Postoji generalizacija Taylorovog teorema za $f(x, y)$ u okolini točke (x_0, y_0). Jasno, i taj teorem se može dalje generalizirati za funkcije od tri i više varijabli, ali to nam za potrebe ove knjige neće trebati.

Označimo sa $L(x_0, y_0; x_1, y_1)$ skup svih točaka na pravcu koji prolazi točkama (x_0, y_0) i (x_1, y_1) i nalaze se između tih točaka (kraće kažemo da je $L(x_0, y_0; x_1, y_1)$ segment između tih dviju točaka).

Teorem 1.1.5. *Neka su $x_0, y_0)$ i $(x_0 + \xi, y_0 + \eta)$ dvije točke i neka je funkcija $f : \mathbf{R}^2 \mapsto \mathbf{R}$ $n+1$ puta neprekidno diferencijabila u nekoj okolini segmenta $L(x_0, y_0; x_0 + \xi, y_0 + \eta)$. Tada je*

$$f(x_0 + \xi, y_0 + \eta) = f(x_0, y_0) + \sum_{j=1}^{n} \frac{1}{j!} \left[\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^j f(x, y) \Big|_{\substack{x=x_0 \\ y=y_0}} + \frac{1}{(n+1)!} \left[\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^{n+1} f(x, y) \Big|_{\substack{x=x_0+\theta\xi \\ y=y_0+\theta\eta}} \quad (1.1.14)$$

za neki $0 \leq \theta \leq 1$. Pritom je $(x_0 + \theta\xi, y_0 + \theta\eta)$ nepoznata točka na segmentu $L(x_0, y_0; x_0 + \xi, y_0 + \eta)$.

Dokaz. Prvo se sjetimo značenja oznake.

$$\left[\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^2 f(x, y) = \xi^2 \frac{\partial^2 f(x, y)}{\partial x^2} + 2\xi\eta \frac{\partial^2 f(x, y)}{\partial x \partial y} + \eta^2 \frac{\partial^2 f(x, y)}{\partial y^2},$$

a slično (po analogiji sa razvojem od $(x + y)^j$) se definira $\left[\xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^j f(x, y)$. Indeksi oblika $x = x_0$ i $y = y_0$ označavaju da se sve prisutne parcijalne derivacije izvrednjavaju u točki (x_0, y_0).

Dokaz relacije (1.1.14) je baziran na Taylorovom teoremu za funkciju jedne varijable $F(t) = f(x_0 + t\xi, y_0 + t\eta)$, $t \in [0, 1]$. Najme, F zadovoljava uvjete teorema 1.1.4., pa je

$$F(1) = f(0) + \frac{F'(0)}{1!} + \cdots + \frac{F^{(n)}(0)}{n!} + \frac{F^{(n+1)}(\theta)}{(n+1)!}$$

za neki $0 \leq \theta \leq 1$. Uočimo da je $F(0) = f(x_0, y_0)$ i $F(1) = f(x_0 + \xi, y_0 + \eta)$. Za prvu derivaciju vrijedi

$$F'(t) = \xi \frac{\partial f(x_0 + t\xi, y_0 + t\eta)}{\partial x} + \eta \frac{\partial f(x_0 + t\xi, y_0 + t\eta)}{\partial y}$$

$$= \left[\xi\frac{\partial}{\partial x} + \eta\frac{\partial}{\partial y}\right] f(x,y) \Big|_{\substack{x=x_0+t\xi \\ y=y_0+t\eta}} .$$

Derivacije višeg reda se dobiju na slični način. ■

Primjer 1.1.5. *Odredimo razvoj funkcije* $f(x,y) = x/y$ *oko točke* $(x_0, y_0) = (2,1)$ *koristeći* $n = 1$. *Imamo* $\xi = x-2$, $\eta = y-1$,

$$\begin{aligned}
\frac{x}{y} &= f(2,1) + \xi\frac{\partial f(2,1)}{\partial x} + \eta\frac{\partial f(2,1)}{\partial y} \\
&\quad + \frac{1}{2}\left[\xi^2\frac{\partial^2 f(x,y)}{\partial x^2} + 2\xi\eta\frac{\partial^2 f(x,y)}{\partial x \partial y} + \eta^2\frac{\partial^2 f(x,y)}{\partial y^2}\right]_{\substack{x=\alpha \\ y=\beta}} \\
&= 2 + (x-2)\cdot 1 + (y-1)\cdot(-2) + \\
&\quad + \frac{1}{2}\left[(x-2)^2\cdot 0 + 2(x-2)(y-1)(\frac{-1}{\beta^2}) + (y-1)^2\frac{2\alpha}{\beta^3}\right] \\
&= x - 2y + 2 - \frac{1}{\beta^2}(x-2)(y-1) + 2\frac{\alpha}{\beta^3}(y-1)^2 ,
\end{aligned}$$

gdje je točka (α, β) *na segmentu* $L(2,1;x,y)$. *Ako je* (x,y) *blizu* $(2,1)$, *tada je segment* $L(2,1;x,y)$ *kratak pa je i* (α,β) *blizu* $(2,1)$ *i vrijedi*

$$\frac{x}{y} \approx x - 2y + 2$$

Graf funkcije $z = x - 2y + 2$ *je ravnina tangencijalna na graf funkcije* $z = x/y$ *u točki* $(x,y,z) = (2,1,2)$. ■

Pretpostavljamo da su čitaoci upoznati sa pojmovima kao što su vektori, matrice, norme, skalarni produkti, a nešto dublje znanje iz linearne algebre koje uključuje osnovne informacije o grupama, vektorskim prostorima, linearnim operatorima, determinantama je dobro došlo za lakše čitanje ove knjige. Pritom je posebno važan pojam linearne nezavisnosti vektora jer se koristi u pojmovima kao što su rang i inverz matrice ili operatora, baze itd.

Za one koji nisu upoznati sa osnovama linearne algebre, uvodima ovdje neke oznake i pojmove da bi lakše razumjeli izreke mnogih tvrdnji koje možda nemaju veze sa linearnom algebrom, ali koriste oznake odatle.

Sa $\mathbf{R}^n$ označavamo skup svih jednostupčanih realnih matrica (koje još zovemo realni vektori)

$$\mathbf{R}^n = \left\{ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \; ; \text{ all } x_i \text{ are real numbers} \right\} .$$

Elemente tog skupa označavamo malim rimskim slovima. Ako su $x, y \in \mathbf{R}$ vektori s komponentama x_i, y_i, $1 \le i \le n$ (kraće pišemo $x = [x_i]$, $x = [x_i]$) i $\alpha \in \mathbf{R}$ realni

broj (skalar), tada se vektori $x+y$ i αx definiraju formulom

$$x+y=\begin{bmatrix} x_1+y_1 \\ \vdots \\ x_n+y_n \end{bmatrix} \quad \text{i} \quad \alpha x=\begin{bmatrix} \alpha x_1 \\ \vdots \\ \alpha x_n \end{bmatrix}.$$

Na isti način se definira $\mathbf{C}^n$ kao skup kompleksnih vektora koji osim realnih mogu imati i kompleksne brojeve kao komponente. Jednako kao gore se definira zbroj kompleksnih vektora i umnožak kompleksnog broja (skalara) i kompleksnog vektora. Uz tako definirane operacije skupovi $\mathbf{R}^n$ i $\mathbf{C}^n$ imaju cijeli niz lijepih svojstava koje ih čine vektorskim prostorima. Ako je $z=[z_i]\in\mathbf{C}$, tada se konjugirano kompleksni vektor $\bar{z}=[\bar{z}_i]$ dobije tako da mu se svaka komponenta kompleksno konjugira. Za realni ili kompleksni vektor $a=[a_i]$, oznaka $|a|=[|a_i|]$ označava apsolutnu vrijednost ili modul vektora a. Dakle, $|a|$ je vektor čije komponente su nenegativni realni brojevi, pa zato vrijede sljedeća jednostavna svojstva: (i) $|a|=0$ ako i samo ako je $a=0$, (ii) $|\alpha a|=|\alpha||a|$, α je skalar, i (iii) $|a+b|\le|a|+|b|$.

Sliča svojstva ima i funkcija norma koja svakom vektoru pridružuje broj. Najpoznatije norme su Euklidska (ili 2-norma) i beskonačna norma. Za $a\in\mathbf{C}$ (ili $\mathbf{R}$) je

$$\|a\|_2=\sqrt{|a_1|^2+\cdots+|a_n|^2}, \quad \|a\|_\infty=\max_{1\le i\le n}|a_i|.$$

Malo općenitije su tzv. p-norme

$$\|a\|_p=\sqrt[p]{|a_1|^p+\cdots+|a_n|^p}, \quad 1\le p\le\infty.$$

Sve te norme zadovoljavaju tri svojstva (definitnost, pozitivna homogenost i nejednakost trokuta) koja funkciju $\|\cdot\|$ čine normom,

(i) $\|a\|=0$ ako i samo ako je $a=0$
(ii) $\|\alpha a\|=|\alpha|\|a\|$ za sve vektore a i skalare α
(iii) $\|a+b\|\le\|a\|+\|b\|$ za sve vektore a i b.

Ista svojstva koja imaju skupovi $\mathbf{R}^n$ i $\mathbf{C}^n$ zajedno sa operacijama zbrajanja i množenja skalarom mogu imati i drugačiji skupovi. Npr. skup

$$C[a,b]=\{f;\ f \text{ neprekidna na } [a,b]\}$$

sa operacijom zbrajanja funkcija $(f+g)(t)=f(t)+g(t)$, $t\in[a,b]$ i množenja funkcija realnim brojem $(\alpha f)(t)=\alpha f(t)$, $t\in[a,b]$ ima ista poželjna svojtva koja ga čine vektorskim prostorom. Tipična norma na tom prostoru je $\|f\|_\infty=\max_{a\le x\le b}|f(t)|$, i jer je norma, zadovoljava svojstva (i)–(iii).

Norme postoje i za matrice. Npr. ako je

$$A==\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

onda je

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \,.$$

Norme na matricama obično zadovoljavaju dodatno svojstvo (konzitentnost) koje se tiče produkta dviju matrica (za vektore x, y produkt xy nije definiran). Za beskonačno normu to svojstvo se zapisuje kao

$$\|AB\|_\infty \le \|A\|_\infty \|B\|_\infty \,, \quad A, B \text{ reda } n \,.$$

Specijalno, vrijedi i

$$\|Ax\|_\infty \le \|A\|_\infty \|x\|_\infty \,,$$

gdje je $x \in \mathbf{R}^n$.

2. GREŠKE U NUMERIČKOM RAČUNANJU

Da bismo mogli ocjenjivati izračunava li neki algoritam, implementiran na računalu kao program, traženo rješenje problema s dovoljnom točnošću potrebno je upoznati pojmove apsolutne i relativne greške, greške unaprijed i unazad, stabilnost algoritma, uvjetovanost problema itd. Svi ti pojmovi nastajali su vremenski postepeno, a uglavnom su vezani uz računanje na elektronskim računalima.

2.1. Tipovi grešaka

U praktičnom računanju postoje dvije osnovne vrste grešaka: greške zbog polaznih aproksimacija i greške zaokruživanja.

2.1.1. Greške zbog polaznih aproksimacija

Taj tip grešaka se često javlja kod rješavanja praktičnih problema. Te greške možemo podijeliti u sljedeće klase: greške modela,greške metode i greške u polaznim podacima.

Greške modela

Najčešće nastaju pojednostavljenjem složenih sustava (u skali od opisa početka svemira do istraživanja atomskih struktura, od složenih reakcija u atomskim centralama do kemijskih i bioloških fenomena u stanicama raka od prognoze vremena do kretanja cijena na burzama) jednostavnijima koje možemo opisati matematičkim zapisom. Često su stvarni fenomeni takovi da ih ne možemo ni približno opisati današnjim matematičkim teorijama ili su previše kompleksni za današnji stupanj matematike. Stoga se vrše razne simplifikacije, s jedne strane da bi uopće opisali fenomene matematičkim zapisom, a s druge strane da bi i njih pojednostavili kako bi ih uopće mogli riješiti. Npr. kod gibanja u zemaljskim uvjetima često se zanemaruje utjecaj otpora zraka. Često se i dobri modeli zamjenjuju slabijim da bi

uopće mogli primijeniti numeričke metode (na primjer, sistemi nelinearnih parcijalnih diferencijalnih jednadžbi se linearizira ju). Pogreške u modelu mogu nastati i kod upotrebe modela u graničnim slučajevima. Na primjer, kod matematičkog njihala se $\sin x$ aproksimira sa x, što vrijedi samo za male kuteve, a upotrebljava se, recimo i za veće kutove nnpr. $15°$. Pogreške modela su neuklonjive, i na korisnicima je da procijene dobivaju li primjenom danog modela očekivane rezultate.

Primjer 2.1.1. *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.

Greške metode

Te greške nastaju kad se beskonačni procesi zamjenjuju konačnim. Također, veličine koje su definirane limesom, poput derivacija i integrala, i rješenja koja su definirana limesima konvergentnih nizova. 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* koje nastaju zamjenom kontinuuma konačnim diskretnim skupom točaka, ili kad se "beskonačno" mala veličina (najčešće u oznaci h ili ε) zamijenjujemo nekim konkretnim malim brojem. Također, kad se derivacija zamijeni podijeljenom razlikom, diferencijalna jednadžba diferencijskom jednadžbom, integral nekom kvadraturnom formulom. Još jednostavniji, tipični primjer greške diskretizacije je aproksimacija funkcije f na intervalu (segmentu) $[a, b]$, vrijednostima te funkcije na konačnom skupu točaka (tzv. mreži) $\{x_1, \ldots, x_n\} \subset [a, b]$.

— *greške odbacivanja* koje nastaju zamjenom beskonačnog niza ili reda konačnim nizom ili sumom (tada odbacujemo ostatak niza ili reda).

Grubo rečeno, diskretizacija je vezana za kontinuum (npr. skupovi $\mathbf{R}$ i $\mathbf{C}$), a odbacivanje za diskretnu beskonačnost ($\mathbf{N}$. $\mathbf{Z}$). Objekti koji nedostaju pri tim zamjenama tvore tip grešaka koji se zovu *greške metode*.

Greške u polaznim podacima

Te greške imaju izvor u mjerenjima fizičkih veličina, u smještavanju podataka u računalo, a također i u prethodnim računanjima. Naime, često su izlazne vrijednosti iz nekog prethodnog računanja, ulazni podaci za novo računanje. Greške od mjerenja ili od smještavanja u računalo je daleko jednostavnije ocijeniti od grešaka koje nastaju uslijed brojnih zaokruživanja u toku računanja.

2.1.2. Greške zaokruživanja

Te greške nastaju u računalima jer oni koriste *konačnu aritmetiku* ili preciznije binarnu *aritmetiku s pomičnom točkom*, kod koje je unaprijed rezerviran određen broj binarnih mjesta za eksponent i za mantisu. Stoga se svaka računska operacija između dva broja izračunava s nekom malom greškom, koja može biti i nula. Ako je nula, tada je rezultat te operacije na danim operandima egzaktano izračunat. Ako nije nula, tada je izračunat s nekom malom greškom koju možemo dosta precizno ocijeniti i koju zovemo greška zaokruživanja.

Ako je algoritam kompleksniji, bit će mnogo računskih operacija. Kod gotovo svake računske operacije postojat će greška zaokruživanja, pa se postavlja pitanje s kojom točnošću ćemo dobiti traženo rješenje? Da bi se dobio odgovor na to pitanje potrebno je istražiti i sam matematički problem i algoritam koji je odabran za računanje rješenja. Pri analizi grešaka koje nastaju u toku računanja koristimo se *teorijom grešaka zaokruživanja*, a osjetljivošću rješenja matematičkog problema koji rješavamo na pomake u polaznim podacima bavi se *teorija perturbacije* za dani problem. Njihovom usklađenom uporabom često je moguće ocijeniti do koje će točnosti algoritam, korišten na računalu, izračunati traženo rješenje. Ako možemo kazati da će ta točnost biti skoro onakva kako to dopušta sam problem zbog nepreciznosti polaznih podataka (koje dolaze npr. od mjernih uređaja ili od smještavanja polaznih podataka u računalo), tada govorimo o stabilnom algoritmu.

Kroz polustoljetni razvoj, teorija grešaka zaokruživanja i teorija stabilnosti numeričkih algoritama ušle su u zrelu fazu, pa je potrebno upoznati osnovne pojmove i rezultate u toj grani numeričke matematike. Cilj ovog poglavlja je savladati tehniku ocjenjivanja grešaka zaokruživanja, ukazati na opasnosti i neželjene efekte koji mogu nastupiti kod računanja na računalima i kalkulatorima, te pokazati kako se dokazuje stabilnost algoritama. Greške zaokruživanja su neizbježne pri svakom zahtjevnijem računanju. One dolaze od korištenja konačne aritmetike u računalima.

Već samo uskladištavanje podataka u računalo dovodi do grešaka jer se svaki broj mora reprezentirati sa konačnim brojem binarnih znamenaka. Zna se da npr. $1/10$ ima u binarnom sustavu prikaz s beskonačno nula i jedinica: $(1/10)_2 = 0.000110011001100110\ldots$ (vidi primjer 2.2.1.). Dakle, ne samo iracionalni, već i mnogi racionalni brojevi, čak i oni umjerene veličine i sa malo dekadskih znamenaka,

ne moraju biti točno reprezentirani u računalu. Kad se jednom polazni brojevi smjeste u računalo, to uglavnom nisu njihove točne već približne binarne reprezentacije. Ako je x realni broj, njegova reprezentacija u računalu se obično označava sa $fl(x)$.

Pretpostavimo sada da su $x = fl(x)$ i $y = fl(y)$ brojevi koji su smješteni u računalo. Tada će svaka aritmetička operacija u računalu između ta dva broja producirati broj koji je najčešće tek aproksimacija točnog rezultata. Tipični slučaj je množenje koje u pravilu daje mantisu rezultata čija je duljina približno zbroj duljina mantisa faktora. Ako smještaj u računalu dopušta p binarnih mjesta za mantisu, tada će svaki od faktora imati mantise duljine p, a umnožak će imati mantisu duljine $2p$ ili $2p-1$. Kako se rezultat opet sprema u memoriju, zadržat će se prvih p znamenaka, a ostale će se odbaciti. Kod zbrajanja i oduzimanja brojeva koji nisu istog reda veličine, u procesu izvršavanja operacije, mantisa jednog od njih (onog s manjom apsolutnom vrijednosti) bit će pomaknuta za jedno ili više mjesta, pa će rezultat opet imati mantisu duljine koja je veća od p. Konačno, kod dijeljenja kao najkompliciranije operacije, algoritam za dijeljenje u računalu također daje rezultat koji nije točan. Detaljno proučavanje (vidjeti npr. [4, 1]) pokazuje da je kod svake operacije u računalu prisutna greška. Ako je greška nula rezultat je točan. Taj zaključak se zapisuje u obliku

$$fl(x \circ y) = (x \circ y)(1+\varepsilon), \quad |\,\varepsilon\,| \leq \epsilon \quad \circ \in \{+,-,*,/\} \tag{2.1.1}$$

pri čemu su $+,-,*,/$ operacije u računalu, a ϵ je tzv. strojna preciznost ili strojni ϵ. Pritom greška ε ovisi o operandima x, y i operaciji $\circ$, dok ϵ ovisi o stroju (ili točnije o IEEE standardu ako ga stroj podržava). Iz zadnje relacije se vidi da je ϵ uniformna gornja ograda za sve greške ε na danom stroju. Kasnije ćemo obrazložiti da za računala koja koriste binarnu aritmetiku sa p binarnih znamenaka u mantisi, vrijedi $\epsilon = 2^{-p+1}$ ili $\epsilon = 2^{-p}$ ovisno o načinu zaokruživanja koje računalo koristi.

Glavna zadaća osobe koja se bavi numeričkim računanjima, je određivanje što bolje aproksimacije rješenja u što kraćem vremenu. Da bi to ostvario, treba pod kontrolom imati sve tipove grešaka koje smo nabrojali, a to znači da mora biti upoznat i sa svim fenomenima koji mogu naići prije i u tijeku računanja, a vezani su uz netočnosti polaznih podataka, međurezultata te konačnog rezultata.

2.1.3. Apsolutna i relativna greška, značajne znamenke

Neka je $\hat{x}$ neka aproksimacija realnog broja x. Najkorisnije mjere za točnost od $\hat{x}$ kao aproksimacije od x su *apsolutna greška*

$$G_{aps}\hat{x} = |\, x - \hat{x}\,|$$

i *relativna greška*

$$G_{rel}\hat{x} = \frac{|\, x - \hat{x}\,|}{|\, x\,|},$$

koja nije definirana za $x = 0$. Drugi način zapisivanja relativne greške je

$$G_{rel}\hat{x} = \mid \rho \mid, \quad \hat{x} = x(1+\rho)\,. \tag{2.1.2}$$

Neki autori izostavljaju u definiciji apsolutne greške znak apsolutne vrijednosti. Time se daje (ili zahtijeva) dodatna informacija o predznaku greške, no tada je pridjev "apsolutna" suvišan. Međutim, često točna vrijednost x nije poznata, već se tek zna neka ograda za $\mid x - \hat{x} \mid$.

Ako je x poznat ili se zna da je "reda veličine jedan", apsolutna greška je dobra mjera za udaljenost aproksimacije od točne vrijednosti. U znanosti međutim, x često varira između vrlo malih (npr. veličine vezane uz atomsku ili molekularnu strukturu) i velikih (npr. veličine vezane uz svemir i njegove objekte) vrijednosti. Tada je relativna greška često primjerenija. Ona ima dodatno lijepo svojstvo da je nezavisna od skaliranja,

$$\frac{\mid x - \hat{x} \mid}{\mid x \mid} = \frac{\mid \alpha x - \alpha\hat{x} \mid}{\mid \alpha x \mid}, \quad \alpha \in \mathbf{R}.$$

Relativna greška je povezana s "brojem točnih značajnih znamenaka". Značajne znamenke u broju su prve netrivijalne (tj. različite od nule) znamenke i one koje slijede iza njih u zapisu. Npr. u broju 6.9990 imamo pet značajnih znamenaka, dok ih u broju 0.0832 imamo tri. Što znači broj "točnih" ili "korektnih" značajnih znamenaka? Npr. ako je

$$\begin{aligned} x &= 1.00000\,, \quad \hat{x} = 1.00499, \quad G_{rel} = 4.99 \cdot 10^{-3}, \\ x &= 9.00000\,, \quad \hat{x} = 8.99899, \quad G_{rel} = 1.12 \cdot 10^{-4}, \end{aligned}$$

vidimo da se $\hat{x}$ slaže s odgovarajućim x na tri značajne znamenke, a ipak je relativna greška za ta dva slučaja različita čak za faktor 44.

Evo jedne moguće definicije: $\hat{x}$ kao aproksimacija od x ima p korektnih značajnih znamenaka ako se $\hat{x}$ i x zaokružuju na isti broj od p značajnih znamenaka. Zaokružiti broj na p značajnih znamenaka znači zamijeniti ga s najbližim brojem koji ima p značajnih znamenaka. Međutim, prema toj definiciji, brojevi $x = 0.9949$ i $\hat{x} = 0.9951$, se ne slažu u dvije značajne znamenke, ali se slažu u jednoj i u tri značajne znamenke. Dakle, definicija nije dobra.

Pokušajmo zato sa ovakvom definicijom: $\hat{x}$ kao aproksimacija od x ima p korektnih značajnih znamenaka ako je $\mid x - \hat{x} \mid <$ jedne polovice jedinice u p-toj značajnoj znamenci od x. Ova definicija implicira da se 0.123 i 0.127 slažu u dvije značajne znamenke, iako će mnogi tvrditi da se slažu u tri.

Vidimo da je relativna greška preciznija mjera od "slaganja u p značajnih znamenaka", pa bi ju trebalo preferirati.

Kad su u pitanju vektori, apsolutna greška od $\hat{x}$ kao aproksimacije od x, definira se pomoću prikladne norme (npr. Euklidske)

$$\|x - \hat{x}\|\,.$$

Relativna greška je

$$\frac{\|x - \hat{x}\|}{\|x\|}.$$

Relacija

$$\frac{\|x - \hat{x}\|}{\|x\|} < \frac{1}{2} \cdot 10^{-p}$$

implicira da komponente $\hat{x}_i$ za koje vrijedi $|x_i| \approx \|x\|$ imaju oko p točnih značajnih znamenaka.

Ako želimo sve komponente od x staviti u isti plan, tada koristimo relativne greške po komponentama, a veličinu

$$\max_i \frac{|\, x_i - \hat{x}_i \,|}{|\, x_i \,|}$$

zovemo (maksimalna) relativna greška po komponentama.

Preciznost i točnost

Ove dvije riječi se često zamijenjuju, pa kad ih već imamo, možemo načiniti sljedeću distinkciju. Točnost neka se odnosi na apsolutnu ili relativnu grešku s kojom se aproksimira neka veličina. Preciznost neka je točnost s kojom se izvršavaju osnovne računske operacije. Kod računanja u aritmetici pomičnog zareza preciznost mjerimo pomoću strojne preciznosti ϵ. Kako je preciznost odrežena brojem bitova u reprezentaciji mantise, ista riječ će se koristiti i za broj bitova u mantisi.

Napomenimo da točnost općenito nije limitirana preciznošću. Naime, pomoću aritmetike dane preciznosti može se simulirati računanje u (proizvoljno) većoj preciznosti. Međutim takove simulacije su toliko skupe (u računskom vremenu) da nisu od praktične važnosti. Stoga pretpostavljamo da se dana konačna aritmetika ne koristi za simulaciju aritmetike veće preciznosti.

2.2. Aritmetika s pomičnom točkom

Na kalkulatorima se često može izabrati tzv. "znanstvena notacija" brojeva, koja npr. broj -27.77 prikazuje kao -2.777×10^1 pri čemu je $-$ *predznak* broja (ili mantise), . je *decimalna točka*, 2.777 je *mantisa*, koja se još zove signifikantni ili razlomljeni dio broja, 10 je *baza* i 1 je *eksponent*. Zapisi -27.77 i -2.777×10^1 su kraće oznake za prikaz broja

$$\begin{aligned} x &= -\left(2 \cdot 10^1 + 7 \cdot 10^0 + 7 \cdot 10^{-1} + 7 \cdot 10^{-2}\right) \\ &= -\left(2 \cdot 10^0 + 7 \cdot 10^{-1} + 7 \cdot 10^{-2} + 7 \cdot 10^{-3}\right) \cdot 10^1, \end{aligned}$$

a kako vrijedi $x = [(-1)\cdot(2\cdot 10^0 + 7\cdot 10^{-1} + 7\cdot 10^{-2} + 7\cdot 10^{-3})]\cdot 10^1$ predznak broja je ujedno predznak mantise.

Općenito, svaki realni broj se može na jednoznačan način zapisati u obliku $\pm m \times 10^e$ gdje je $1 \leq m < 10$.

Računala koriste sličnu reprezentaciju broja koja se zove *pomična točka*, ali se ne uzima baza 10 već baza 2 (s iznimkom baze 16 kod računala IBM 370 i baze 10 kod većine kalkulatora). Tako je npr.

$$\begin{aligned} y &= (11.1011)_2 = 1\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} + 1\cdot 2^{-4} \\ &= \left(1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4} + 1\cdot 2^{-5}\right)\cdot 2^1 \\ &= (1.11011)_2\cdot 2^1\,, \end{aligned}$$

Ovdje je $(11.1011)_2$ binarna reprezentacija broja y, a njen oblik u prikazu s pomičnim točkom $(1.11011)_2\cdot 2^1$. Lako se izračuna da y ima decimalnu reprezentaciju $(3.625)_{10}$, a jer smo mi navikli na decimalni sistem, kažemo da je y broj 3.625.

2.2.1. Pretvaranje decimalne u binarnu reprezentaciju

Ne samo iracionalni, već i mnogi racionalni brojevi imaju beskonačno mnogo znamenaka u decimaknom brojnom sustavu. Npr. 1/3 ima prikaz $0.33333\cdots$ 5/17 ima prikaz 0.2941176470588235 2941176470588235 2941176470588235... Uočimo da se i svaki konačni prikaz broja može napisati s beskonačno znamenaka, npr. $x = 32.75 = 32.74999999\cdots$. Slična je situacija i s brojevima zapisanim u binarnom sustavu. Međutim, zanimljivo je da mnogi racionalni brojevi imaju u dekadskom sustavu konačni prikaz, dok u binarnom sustavu imaju beskonačni prikaz. Da li vrijedi i obrat?

Da bismo se u to uvjerili, izvedimo prvo algoritam za pretvaranje decimalnog oblika u binarni oblik.

Neka x prvo ima konačni binarni prikaz. Tada x možemo prikazati kao sumu njegovog cijelog i razlomljenog dijela,

$$\begin{aligned} x &= (a_k a_{k-1}\cdots a_1 a_0.b_1 b_2\cdots b_{l-1}b_l)_2 = x_c + x_r \\ x_c &= (a_k a_{k-1}\cdots a_1 a_0)_2 = a_k\,2^k + a_{k-1}\,2^{k-1} + \cdots + a_1\,2^1 + a_0\,2^0 \qquad (2.2.1) \\ x_r &= (.b_1 b_2\cdots b_{l-1}b_l)_2 = b_1\,2^{-1} + b_2\,2^{-2} + \cdots + b_{l-1}\,2^{-(l-1)} + b_l\,2^{-l} \qquad (2.2.2) \end{aligned}$$

Ako je zadan cijeli broj x_c u decimalnom obliku, kako odrediti njegove binarne znamenke $a_k,\ldots,a_0$, koristeći nama razumljivu decimalnu aritmetiku ?

Iz relacije (2.2.1) vidimo da cjelobrojno dijeljenje broja x_c sa 2 daje novi cijeli broj $x_c' = a_k 2^{k-1} + a_{k-1}2^{k-2} + \cdots + a_1 2^0$ i ostatak a_0. Ako ponovimo postupak

na broju x_c' dobijemo sljedeću znamenku a_1. Ponavljanjem tog postupka (u našem slučaju k puta) dobivamo sve binarne znameke broja x_c u redosljedu od a_0 do a_k.

S obzirom da je x_c cijeli broj, on će i u binarnom sustavu imati konačni prikaz. Pritom će binarni prikaz imati $[log_2(x_c)] + 1$ binarnih znamenaka (bitova). Oznaka $[z]$ za realni broj z označava najveći cijeli broj koji je $\leq z$. Ako x_c ima p decimalnih znamenaka (decimala), tj. $10^{p-1} \leq x_c < 10^p$ vrijedit će zbog monotonosti logaritamske funkcije

$$\log_2(10^{p-1}) \leq log_2(x_c) < \log_2(10^p) \text{ ili } (p-1)\log_2(10) \leq log_2(x_c) < p\log_2(10).$$

Kako je $\log_2(10) \approx 3.3219$, binarna reprezentacija zahtijeva oko $3.3p$ znamenka, dakle oko 3.3 puta više nego decimalna reprezentacija.

Neka je sada x_r kao u relaciji (2.2.2). Tada množenjem broja x_r s dva i uzimanjem cijelog dijela rezultata dobivamo b_1. Radeći isti postupak s ostatkom $x_r' = b_2 2^{-1} + \cdots + b_{l-1}2^{-(l-2)} + b_l 2^{-(l-1)}$, dobivamo binarnu znamenku b_2. Nastavljajući postupak (l puta) lako dobivamo sve binarne znamenke broja x_r u redosljedu od b_1 do b_l. Zbog jednostavnosti postupka, odmah zaključujemo da postupak vrijedi i u slučaju kad x_r nema konačni binarni prikaz, ali tada imamo beskonačno koraka.

U sljedećem algoritmu za pretvaranje decimalne reprezentacije brojeva u binarnu, pretpostavljamo korištenje decimalne aritmetike.

Algoritam 2.2.1. *Neka je zadan racionalni broj $x = x_c + x_r$ pri čemu je x_c cijeli dio broja x, a x_r razlomljeni dio tako da je $0 \leq x_r < 1$. Algoritam računa binarnu reprezentaciju decimalnih brojeva x_c i x_r. Algoritam koristi operaciju* `div` *za cjelobrojno dijeljenje,* `mod` *za dobivanje ostatka kod cjelobrojnog dijeljenja i* `int` *za zaokruživanje realnog broja do cijelog broja u smjeru nule.*

```
y:=x_c;
k:=-1;
repeat
 k:=k+1
 a(k):=mod(y,2);
 y:=div(y,2);
until y=0
```

```
z:=x_r;
l:=0;
repeat
 l:=l+1;
 zz:=2*z;
 b(k):=int(zz);
 z:=zz-b(k);
until z=0 or l > lmax
```

Pritom `lmax` označava najveći dopušteni broj binarnih znamenaka u binarnoj mantisi (sjetimo se da 1/10 ima beskonačno nula i jedinica).

Primjer 2.2.1. *Koristeći algoritam 2.2.1. pretvorit ćemo decimalni broj* 111.1 *u binarni. Kad radimo "ručno" zgodno je nacrtati vodoravnu crtu pa kod pretvaranja broja* 111 *za svaki korak pišemo s desne strane rezultat cijelobrojnog dijeljenja, a ispod dividenda ostatak kod dijeljenja. Točka na kraju označava kraj.*

111	*55*	*27*	*13*	*6*	*3*	*1*	*0*
1	*1*	*1*	*1*	*0*	*1*	*1*	.

što daje $111 = (110111)_2$

Slično, kod pretvaranja decimalnog dijela broja, 0.1 *u binarni broj, s desne strane pišemo rezultat množenja s dva, a s ispod crte rezultat zaokruživanja do cijelog broja u smjeru nule. Ako je rezultat dijeljenja veći ili jednak jedan, u sljedećem množenju, prije samog množenja automatski oduzimamo jedinicu. Binarna točka prethodi računanju i pišemo ju ispod polaznog decimalnog broja.*

0.1	*0.2*	*0.4*	*0.8*	*1.6*	*1.2*	*0.4*	*0.8*	*1.6*	*1.2*
.	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*

pa se dobiva $0.1 = (.0001100110011001100\ldots)_2$, *odnosno*

$$(111.1)_{10} = (1101111.0001100110011001100\ldots)_2.$$

■

2.2.2. Strojna reprezentacija brojeva

U programskim jezicima postoji nekoliko vrsta aritmetika koje koriste posve određene tipove podataka. Cjelobrojna aritmetika koristi *cjelobrojni tip* podataka koji čine konačni interval u skupu cijelih brojeva, realna aritmetika koristi takozvani *realni tip* podataka kojem pripada tek konačni podskup racionalnih brojeva. Nešto veći skup racionalnih brojeva čini *tip brojeva u dvostrukoj preciznosti* čije mantise su više nego dvostruko dulje od mantisa realnog tipa, a i pripadna aritmetika je drukčija od realne aritmetike (jer više nego dvostruko preziznija). Konačno, konačni podskup skupa kompleksnih brojeva je reprezentiran tipom kompleksnih brojeva (koji nije prisutan kod svih programskih jezika) nad kojim je definirana pripadna kompleksna aritmetika. Svaki od tih glavnih tipova obično ima svoje podtipove (ili ekstenzije) koje obično određuje broj bajtova. Jedan bajt (engl. byte) čini 8 bitova, koji su dostatni za reprezentaciju jednog znaka. Današnja računala imaju ćelije od 32 bita pa su tome prilagođeni građa procesora, implementacije aritmetika kao i cijeli operacioni sustav. Nakon 2002. očekuju se računala bazirana na 64 bitnim ćelijama.

Strojna reprezentacija cijelih brojeva Cijeli pozitivni brojevi se reprezentiraju u 32 bitnoj čeliji kao desno pozicionirani binarni brojevi. Npr. broj 111 će biti smješten ovako

00000000000000000000000001101111

Na taj način možemo smjestiti sve brojeve od 0 (reprezentirane s 32 nule) do $2^{32}-1$ (reprezentirane s 32 jedinice). Broj 2^{32} je prevelik njegova binarna reprezentacija zahtijeva jednu jedinicu i 32 nule. Neki programski jezici imaju tip podataka *cijeli broj bez predznaka* i spomenuta reprezentacija je odgovarajuća za njih.

Međutim, ako unutar 32 bita moramo moći spremiti i pozitivne i negativne brojeve, prvo moramo vidjeti kako spremiti negativne cijele brojeve. Najočitija mogućanost je uzeti jedan bit za predznak, na primjer prvi, tako da je taj bit 0 ako je broj pozitivan i 1 ako je negativan. S obzirom da na raspolaganju imamo još 31 bit, raspon tako reprezentiranih brojeva je od od $-2^{31}-1$ do $2^{31}-1$. Međutim, gotovo sva današnja računala koriste pametniji način reprezentacije negativnih brojeva koji se zove *drugi komplement*[1] i piše 2. komplement. U sustavu s drugim komplementom, nenegativni cijeli broj x, $0 \leq x \leq 2^{31}-1$ se smještavaju kao binarna reprezentacija tog broja, dok se $-x$, $1 \leq x \leq 2^{31}-1$ smještava kao binarna reprezentacija broja $2^{32}-x$. Npr. broj -111 se smještava kao

11111111111111111111111110010000

Primjer 2.2.2. *Zbrojimo u binarnoj aritmetici brojeve* 111 *i* -111 *koristeći drugi komplement. Možemo koristiti kao i računala 32 bitne reprezentacije. Imamo*

	00000000000000000000000001101111
+	11111111111111111111111110010000
1	00000000000000000000000000000000

Ovdje je 1 *tzv. prekobrojni bit jer je na 33. mjestu i on se odbacuje. Preostaje* 00000000000000000000000000000000 *što je reprezentacija broja nula.* ■

Zadatak 2.2.1. *Koliko se različitih cijelih brojeva može reprezentirati ako se koristi sustav*
(a) predznak i modul i
(b) 2. komplement?
Koristite potencije od 2. *Za koji od ovih sustava je reprezentacija nule jedinstvena? Promotrite slučajeve kad za reprezentaciju cijelih brojeva imate na raspolaganju*
(i) 32 bita, tj. 4 bytea
(ii) 16 bita, tj. 2 bytea
(iii) 8 bita, tj. 1 byte
(iv) 64 bita, tj. 16 bytea.

Zadatak 2.2.2. *Pokažite da je u sustavu reprezentacije cijelih brojeva pomoću 2. komplementa (npr. u 32 bitnom formatu), najljeviji bit* 1 *ako i samo ako je* x *negativan.* ■

Zadatak 2.2.3. *Da bi u sustavu reprezentacije cijelih brojeva pomoću 2. komplementa sa 32 bita, od već uskladištenog broja* x *dobili* $-x$ *treba poduzeti dva koraka. Prvi je promijeniti svaku* 0 *u* 1 *i svaki* 1 *u* 0. *Koji je drugi korak?* ■

[1]Ime dolazi odatle što se je između 1960. i 1980. na nekim superračunalima koristio tzv. 1. komplement kod kojeg se negativni broj $-x$ smjestio kao binarna reprezentacija od $2^{32}-x-1$.

Sva računala imaju ugrađene hardwareske instrukcije za zbrajanje cijelih brojeva. Ako se zbroje dva cijela pozitivna broja, rezultat može biti broj koji je veći od maksimalnog prikazivog broja. U tom slučaju dolazi do tzv. *overflow*a. Što računalo u takvom slučaju načini ovisi o kompileru, najčešće prekine računanje uz adekvatnu poruku. Do prekoračenja može doći i ako se zbrajaju dva negativna broja. Ako se zbrajaju dva broja sa suprotnim predznakom, do prekoraćenja neće doći, iako može doći do jednog suvišnog bita (kao kod zbroja $x + (-x)$).

Promotrimo operaciju $x + (-y)$, gdje su $0 \leq x \leq 2^{31} - 1$ i $0 \leq y \leq 2^{31}$. S obzirom da je broj $-y$ u reprezentaciji 2. komplementa binarni prikaz od $2^{32} - y$, zbroj ćemo zapisati kao $2^{32} + x - y = 2^{32} - (y - x)$. Ako je $x \geq y$ najljeviji bit u reprezentaciji od $2^{32} + (x - y)$ će biti jedinica (na poziciji potencije 2^{32}) i ona će se odbaciti, pa će ostati točan rezultat $x - y$. Ako je $x < y$, rezultat $2^{32} - (y - x)$ stane unutar 32 bitne reprezentacije i reprezentira broj $-(y - x)$. Ovo razmatranje pokazuje jedno važno svojstvo sustava reprezentacije s 2. komplementom: nije potrebna nikakva posebna hardwareska operacija za cjelobrojno oduzimanje. Jer, kad se jednom broj $-y$ reprezentira, dovoljno je koristiti samo operaciju hardwareskog zbrajanja.

Zadatak 2.2.4. *Pokažite detalje razmatranja za cjelobrojnu sumu* $50 + (-200)$, $200 + (-50)$ *i* $200 + 200$, *koristeći 8 bitni format.* ■

Pored zbrajanja, postoje još dvije standardne hardwareske operacije za cjelobrojne operande: cjelobrojno množenje i cjelobrojno dijeljenje. Množenje može lako dovesti prekoraćenja. Kod dijeljenja jedino ako je divizor nula. U oba slučaja se računanje prekida uz odgovarajuću poruku.

Strojna reprezentacija realnih brojeva U računalu se realni brojevi reprezentiraju pomoću binarne reprezentacije u znanstvenoj notaciji,

$$x = \pm m \times 2^e, \quad \text{gdje je } 1 \leq m < 2\,. \tag{2.2.3}$$

Stoga je

$$m = (b_0.b_1b_2b_3\ldots)_2 \quad \text{pri čemu je } b_0 = 1\,. \tag{2.2.4}$$

Na primjer, broj $111.5 = (1101111.1)_2$ se može napisati kao $(1.1011111)_2 \times 2^6$. Vidimo da je u znanstvenom prikazu binarna točka pomaknuta za 6 mjesta u lijevo, ali se se pritom eksponent povećao za 6. Kako se zahtijeva da je $b_0 = 1$, možemo pisati

$$m = (1.b_1b_2b_3\ldots)_2\,.$$

U tom prikazu, binarne znamenke desno od binarne točke čine razlomljeni dio mantise, a relacije (2.2.3) i (2.2.4) predstavljaju *normalizirani* oblik ili reprezentaciju broja x. Sam proces dobivanja tog oblika se zove normalizacija.

Da bi spremili normalizirane brojeve u računalo, podijelimo memorijsku riječ (sadržaj jedne ćelije) u tri dijela koja zovemo polja. Kod 32-bitnih mašina, riječ ima

32 bita pa se obično dijeli na sljedeći način: 1 bit za predznak, 8 bitova za eksponent e i 23 bitova za mantisu. Bit za predznak je 0 (1) ako je broj pozitivan (negativan). Polje za eksponent ima osam bitova pa može reprezentirati ekponent e između granica -128 i 127 (npr. pomoću reprezentacije s dvostrukim komplementom). Preostala 23 bita za smještaj mantise se zapravo koriste za smještaj razlomljenog dijela mantise, jer je uvijek $b_0 = 1$ pa ga ne treba spremati. Realni broj x nazivamo *egzaktno reprezentabilnim* u računalu ili *brojem s pomičnom točkom* ako se na opisani način može bez greške smjestiti u računalo. Ako broj nije egzaktno reprezentabilan u računalu, on se mora prije smještavanja u računalu zaokružiti.

Primjer 2.2.3. *U primjeru 2.2.1. je pokazano da* $(111.1)_{10} = (1101111.000110011\ 00110011\ldots)_2$, *pa* $x = 111.1$ *nije egzaktno reprezentabilan u računalu. Ako pišemo* $x = (1.10111100011001100110011001100\ldots)_2 \cdot 2^6$, *tada se* x *može smjestiti u računalo, nakon zaokruživanja na sljedeći način*

$$\boxed{0\,|\ eksp_2(6)\ |10111100011001100110011}$$

pri čemu $eksp_2(6)$ *označava da na tom mjestu dolazi binarna reprezentacija broja* 6. *S druge strane broj* $y = -2^{22} = -4194304$, *će imati egzaktnu reprezentaciju*

$$\boxed{1\,|\ eksp_2(22)\ |00000000000000000000000}$$

■

Zadatak 2.2.5. *Koji su najveći i najmanji pozitivni brojevi s pomičnom točkom u opisanom formatu? Ne zaboravite na skriveni bit i na ograničen interval* $[-128, 127]$ *za eksponent. Koji je najmanji cijeli pozitivni broj koji nije egzaktno reprezentabilan?*

Zadatak 2.2.6. *Pretpostavimo da umjesto oblika (2.2.4) u relaciji (2.2.3) zahtijevamo* $S = (0.b_1b_2b_3\ldots)_2$ *pri čemu je* $b_1 = 1$. *Pretpostavimo da se u polje za mantisu smjeste binarne znamenke* $b_2b_3\ldots b_{24}$ *i da je kao i prije raspon za eksponent* $-128 \leq e \leq 127$. *Koji je najveći i najmanji pozitivni broj u tom formatu? koji je najmanji pozitivni broj koji nije egzaktno reprezentabilan?*

Strojni epsilon, ulp i preciznost Preciznost p možemo u danom brojevnom sistemu definirati brojem bitova u mantisi pri čemu se računa i skriveni bit. U opisanom brojevnom sustavu je $p = 24$. U brojevnom sustavu s preciznošću p, normalizirani broj s pomičnom točkom ima oblik

$$x = \pm(1.b_1b_2\ldots b_{p-2}b_{p-1}) \times 2^e\,. \tag{2.2.5}$$

Najmanji takav x koji je veći od 1 je $(1.00\ldots01)_2 = 1 + 2^{-(p-1)}$. Razmak između ta dva broja se zove *strojni* ili *mašinski* epsilon i pišemo

$$\epsilon_M = 2^{-(p-1)}\,.$$

Općenitije, za x kao u relaciji (2.2.5) se definira

$$\text{ulp}(x) = (0.00\ldots01)_2 \times 2^e = 2^{-(p-1)} \times 2^e = \epsilon_M \times 2^e\,. \tag{2.2.6}$$

Ulp je kratica engleskih riječi *unit in the last place*. Ako je $x > 0$ ($x < 0$), ulp(x) je razmak između x i sljedećeg većeg (manjeg) reprezentabilnog broja.

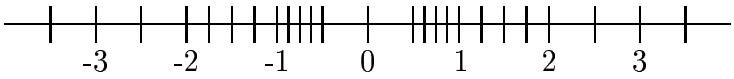
Zadatak 2.2.7. *Neka je $p = 24$, tako da je $\epsilon_M = 2^{-23}$. Odredi ulp(x) za $x = 0.5$, 0.125, 3, 8, 10, 100, 125.* ■

Još se postavlja pitanje kako spremiti nulu u računalo? To se postiže korištenjem specijalnog niza bitova u polju za eksponent. O tomu ćemo više reći u točki s opisom IEEE standarda.

Da bismo dobili ideju koje točke na brojevnom pravcu odgovaraju reprezentabilnim brojevima, definirat ćemo pojednostavljeni *brojevni sustav za prikazivanje* ili kraće BSP, koji se sastoji od brojeva oblika

$$\pm(b_0.b_1b_2)_2 \times 2^e, \quad b_0, b_1, b_2 \in \{0, 1\}, \quad e \in \{-1, 0, 1\}.$$

Preciznost je $p = 3$, najveći prikaziv broj je $(1.11)_2 \times 2^1 = (3.5)_{10}$, a najmanji $(1.00)_2 \times 2^{-1} = (0.5)_{10}$. Kako je desni susjed od 1 u BSP 1.25, strojni epsilon je $\epsilon_M = 0.25$. Ako promatrimo sve brojeve u BSP za koje je $e = 0$, vidjet ćemo da su to brojevi 1, 1.25, 1.5 i 1.75. Između svih njih je isti razmak $\text{ulp}(x) = \epsilon_M$. Brojevi u BSP za koje je $e = 1$ dobiju se množenjem s dva onih brojeva za koje je $e = 0$, pa su to brojevi 2, 2.5, 3, 3.5 i za njih je $\text{ulp}(x) = 2\epsilon_M$. Slično, brojevi iz BSP za koje je $e = -1$ su 0.5, 0.625, 0.75 i 0.825 i za njih je $\text{ulp}(x) = \epsilon_M/2$. Vidimo da je razmak između broja $x \in$BSP i njegovog desnog susjeda u BSP jednak $\text{ulp}(x) = \epsilon_M \times 2^e$.



Slika 2.2.1. BSP brojevni sustav

Uočimo da je razmak između nule i ± 0.5 mnogo veći od razmaka brojeva između ± 0.5 i ± 1. Uskoro ćemo vidjeti da se razmak između nule i ± 0.5 može ispuniti tzv. *subnormalnim* brojevima.

2.3. IEEE Aritmetika

U kasnim 70tim i ranim 80tim godinama, došlo je do izuzetne suradnje eksperata iz industrije i sa sveučilšta. Predvođeni W. Kahanom sa Californijskog sveučilišta

u Berkeleyu stručnjaci iz računarstva i dizajneri mikročipova uspjeli su se dogovoriti oko standarda smještavanja brojeva u računalo i standarda za aritmetiku u računalu. To je bilo vrijeme naglog razvoja osobnih računala s Intelom i Motorolom kao glavnim proizvođačima procesora. IEEE standard za binarnu aritmetiku je publiciran 1985. i spominje se kao IEEE 754. Razvoj kalkulatora i malih, tzv. handheld računala koja koriste decimalnu aritmetiku utjecao je na razvoj standarda IEEE 854 koji vrijedi za svaku bazu i konzistentan je sa prethodnim standardom za binarnu aritmetiku. Kad se spominje "IEEE aritmetika" podrazumijeva se da ta aritmetika udovoljava spomenute standarde.

Bitni zahtjevi IEEE standarda su: (i) konzistentna reprezentacija brojeva s plivajućom točkom na svim strojevima koji prihvaćaju standard, (ii) korektno zaokružene računske operacije u svim načinima rada i (iii) konzistentno tretiranje izvanrednih situacija kao npr. dijeljenje s nulom. U tom standardu je vodeća jedinica normalizirarog broja skrivena, pa je zato potrebna specijalna reprezentacija broja nula. Uvodi se i specijalna reprezentacija brojeva $\pm\infty$, brojeva ± 0 (to je za razliku od zamišljenih različitih brojeva $+\infty$ i $-\infty$ isti broj), te specijalnih izmišljenih brojeva "NaN" ("Not a Number" koji su oznaka za npr. kvocijente $0/0$). IEEE format specificira dva osnovna formata: jednostruki i dvostruki.

2.3.1. Jednostruki format

Ovaj format je opisan tabelom 2. Pritom znak $\pm$ u nizu bitova znači 0 ako je prdznak broja $+$ i 1 ako je $-$. Iz prvog reda vidimo da je nula reprezentirana s

$$\boxed{\pm \mid 00000000 \mid 00000000000000000000000}.$$

Svi retci u tabeli 2 osim prvog i zadnjeg reprezentiraju normalizirane brojeve. Vidimo da eksponent nije prikazan ni kao drugi komplement niti u obliku predznak plus modul, već kao $127 + e$. Broj $127 + e$ ćemo zvati karakteristika. Npr. broj 1 se reprezentira kao

$$\boxed{0 \mid 01111111 \mid 00000000000000000000000},$$

dok se broj $0.1 = (1.100110011\ldots)_2 \times 2^{-4}$ reprezentira kao

$$\boxed{0 \mid 01111011 \mid 10011001100110011001100}.$$

Najmanji i najveći eksponent od 2 u tom formatu su $e_{min} = -126$ i $e_{max} = 127$ Najmanji normalizirani broj $N_{min} = (1.000\ldots0)_2 \times 2^{-126} = 2^{-126} \approx 1.1755 \times 10^{-38}$ se reprezentira s

$$\boxed{\pm \mid 00000001 \mid 00000000000000000000000}.$$

$$\boxed{\pm \mid a_1 a_2 \ldots a_8 \mid b_1 b_2 \ldots b_{23}}$$

akoje niz $a_1 a_2 \ldots a_8$	onda je numerička vrijednost
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 \ldots b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{-125}$
$(00000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{-124}$
$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111100)_2 = (252)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{125}$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ ako su svi $b_i = 0$, inače NaN

Tablica 2.3.1. IEEE jednostruki format

dok se najveći broj $N_{max} = (1.111\ldots1)_2 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4028 \times 10^{38}$ reprezentira s

$$\boxed{\pm \mid 11111110 \mid 11111111111111111111111}.$$

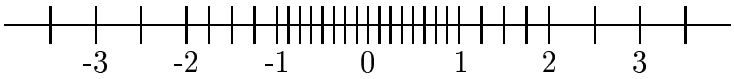
Odmah vidimo da je $1/N_{min} < N_{max}$, pa inverz najmanjeg normaliziranog broja ne dovodi do prekoračenja gornje granice (overflow).

Iz zadnjeg retka tabele vidimo da niz bitova 1111111 u eksponencijalnom dijelu reprezentacije vodi ili do $\pm\infty$ (samo ako su svi b_i jednaki 0) ili do NaN.

Konačno, pogledajmo još jednom prvi redak tabele 2. U njemu je eksponencijalni dio uvijek 0000000 dok razlomljeni dio može imati bilo koji izbor jedinica i nula. Ako su svi b_i nula, imat ćemo $+0$ ili -0 ovisno o prvom bitu. Ako nisu svi b_i nula, dobit ćemo tzv. *subnormalne* ili *denormalizirane* brojeve koji ekvidistantno ispunjuju razmak između $-N_{min}$ i $+N_{min}$. Najmanji pozitivni subnormalni broj je reprezentiran kao

$$\boxed{\pm \mid 00000001 \mid 00000000000000000000001},$$

a vrijednost mu je $2^{-149} = (0.000\ldots01) \times 2^{-126}$. To je najmanji broj koji se može reprezentirati pomoću jednostrukog formata. Najveći subnormalni broj je $(0.111\ldots11)_2 \times 2^{-126}$ i on je za 2^{-149} manji od N_{min}. Sve reprezentabilne brojeve u brojevnom sustavu za prikazivanje, uključujući i subnormalne brojeve sada možemo prikazati slikom 2



Slika 2.3.1. BSP brojevni sustav sa subnormalnim brojevima

Subnormalni brojevi su manje točni od normaliziranih. Npr. $(1/10) \times 2^{-136} = (0.11001100\ldots)_2 \times 2^{-139}$ ima reprezentaciju

$$\boxed{0|\ 00000000\ |\ 00000000000001100110011}.$$

Zadatak 2.3.1. *Odredite IEEE jednostruki format za reprezentaciju sljedećih brojeva s pomičnom točkom:* 3, 2000, 11.5, 11.5×2^{100} *i* 0.1×2^{-142}. ■

Zadatak 2.3.2. *Pokušajte napisati algoritam koji određuje koji je od dva broja u jednostrukom IEEE formatu x i y veći od drugoga. Treba uspoređivati bitove s lijeva na desno i donijeti zaključak na prvom mjestu na kojem se razlikuju. Činjenica da se to može učiniti tako jednostavno je utjecala na to da se eksponent smjesti u polje pomoću karakteristike.* ■

2.3.2. Dvostruki format

Kod zahtijevnijih računanja, jednostruki format nije adekvatan, bilo zbog neizbježnih grešaka zaokruživanja, bilo zbog preuskog raspona brojeva u tom formatu. Zato IEEE standard specificira i drugi tzv. dvostruki format koji koristi riječ od 64 bita. Detalji su vidljivi u tabeli 2.

Ideje su iste, samo su polja za mantisu i eksponent veća: 11 bita za eksponent i 52 bita za razlomljeni dio mantise, pa su zato $e_{min} = -1022$ i $e_{max} = 1023$, te $N_{min} = 2^{-1022} =\approx 2.225 \times 10^{-308}$ i $N_{max} = (2 - 2^{-52}) \times 2^{1023} \approx 1.797693 \times 10^{308}$.

IEEE standard zahtijeva da stroj omogućava jednostruki format. Dvostruki format se zahtijeva tek kao mogućnost, iako ga gotovo sva računala koja podržavaju standard imaju. Podrška na zahtjeve standarda može biti programska ("soft-

wareska") ili elektronička ("hardwareska"), iako zbog brzine rada računala, proizvođaći računala najčešće daju hardwaresku podršku standardu.

$$\boxed{\pm \mid a_1 a_2 \dots a_{11} \mid b_1 b_2 \dots b_{52}}$$

akoje niz $a_1 a_2 \dots a_8$	onda je numerička vrijednost
$(00000000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$
$(00000000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{-1021}$
$(00000000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{-1020}$
$\vdots$	$\vdots$
$(01111111111)_2 = (1023)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{0}$
$(10000000000)_2 = (1024)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111111100)_2 = (2044)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1021}$
$(11111111101)_2 = (2045)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1022}$
$(11111111110)_2 = (2046)_{10}$	$\pm(1.b_1 b_2 \dots b_{52})_2 \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$\pm\infty$ ako su svi $b_i = 0$, inače NaN

Tablica 2.3.2. IEEE dvostruki format

Sljedeće dvije tabele prikazuju karakteristične podatke za jednostruki i dvostruki format. Standard također snažno preporuča podršku za tzv. proširени (extended) format koji bi trebao imati barem 15 bitova za eksponent i barem 63 bita za mantisu. Intelovi mikroprocesori implementiraju aritmetiku s proširenim formatom u hardwareu koristeći 80 bitne registre od čega se 15 koristi za eksponent i 64 bita za mantisu, pri čemu za razliku od jednostrukog i dvostrukog formata, vodeći bit (jedinica) nije skriven. Drugi strojevi (npr. Sun i Sparc) implementiraju prošireni format sa 128 bita, ali softwareski, pa je aritmetika ovdje sporija. Kod intelovog

format	e_{min}	e_{max}	N_{min}	N_{max}
jednostruki	-126	127	$2^{-126} \approx 1.2 \times 10^{-38}$	$\approx 2^{128} \approx 3.4 \times 10^{38}$
dvostruki	-1022	1023	$2^{-1022} \approx 2.2 \times 10^{-308}$	$\approx 2^{1024} \approx 31.8 \times 10^{308}$

Tablica 2.3.3. Raspon brojeva s pomičnom točkom u IEEE formatima

proširenog formata, prvi desni susjed od 1 bi bio $1 + 2^{-64}$, ali kako se on ne može

reprezentirati jer nema skrivenog bita, prvi desni susjed je $1 + 2^{-63}$. S obzirom da je $\log_{10}(2^{23}) \approx 6.9236$, $\log_{10}(2^{24}) \approx 7.2247$, preciznost $p = 24$ odgovara približno 7 značajnih decimalnih znamenki. Slično, preciznost $p = 53$ odgovara približno 16, a $p = 64$ odgovara približno 19 značajnih decimalnih znamenki. Moderna računala

format	preziznost	strojni epsilon
jednostruki	$p = 24$	$\epsilon_M = 2^{-23} \approx 1.2 \times 10^{-7}$
dvostruki	$p = 53$	$\epsilon_M = 2^{-52} \approx 2.2 \times 10^{-16}$
prošireni (Intel)	$p = 64$	$\epsilon_M = 2^{-63} \approx 1.1 \times 10^{-19}$

Tablica 2.3.4. Preciznost kod IEEE formata

adresiraju memoriju po byteovima, pa 32 bitnu riječ adresiraju sa 4 bytea, nazovimo ih $B_1,\ldots,B_4$, pri čemu je $B_4 = B_1 + 3$. U jednostrukom formatu, najvažniji je byte u kojem su smješteni $\sigma, a_1, \ldots a_7$ (σ je predznak). Ako je taj byte adresiran s B_1 (B_4), tada se takav adresni sustav zove Big (Small) Endian. Npr. IBM i Sun koriste BIG, dok Intel koristi Small Endian (neki procesori kao DEC Alpha mogu raditi s oba sustava). To znači da kod transfera podataka s jedne mašine na drugu traba biti oprezan.

2.3.3. Zaokruživanje u BSPT

Brojevni sustav s pomičnom točkom ćemo kraće označavati s BSPT, a skup brojeva s pomičnom točkom s BPT. Te oznake se odnose na razmatrani sustav/skup brojeva za koji ćemo smatrati, ako drugačije ne naznačimo, da zadovoljavaju IEEE standard.

Za realni broj x ćemo reći da leži u *intervalu normaliziranih brojeva* danog sustava s pomičnom točkom, ako vrijedi

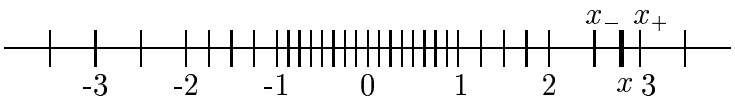
$$N_{min} \leq \mid x \mid \leq N_{max}\,.$$

Dakle brojevi ± 0, $\pm\infty$ i subnormalni brojevi nisu u tom intervalu, iako pripadaju spomenutom brojnom sustavu.

Neka je x realni broj koji nije reprezentabilan u sustavu brojeva s pomičnom točkom. Tada je barem jedna od sljedećih tvrdnji istinita: x leži izvan intervala normaliziranih brojeva (npr. $x = 2^{129}$) i binarna reprezentacija od x zahtijeva više od p bitova za egzaktnu reprezentaciju (npr. $x = 0.1$). U oba slučaja porebno je zamijeniti x s brojem iz sustava brojeva s pomičnom točkom. Neka je $x_- \leq x \leq x_+$ pri čemu su x_- i x_+ najbliži brojevi s pomičnom točkom broju x (vidi sliku 3).

Ako pretpostavimo da je

$$x = (1.b_1 b_2 \ldots b_{p-1} b_p b_{p+1} \ldots)_2 \times 2^e \tag{2.3.1}$$



Slika 2.3.2. Zaokruživanje u BSPT brojevnom sustavu

onda su

$$\begin{aligned} x_- &= (1.b_1b_2\ldots b_{p-1})_2 \times 2^e \\ x_+ &= [(1.b_1b_2\ldots b_{p-1})_2 + (0.00\ldots 01)_2] \times 2^e \end{aligned}$$

i razmak između x_- i x_+ je $2^{p-1}\times 2^e = \text{ulp}(x_-)$. Ako je $x > N_{max}$, tada je $x_- = N_{max}$ i $x_+ = \infty$. Ako je $0 < x < N_{min}$, tada je x_- ili nula ili subnormalni broj, a x_+ je subnormalni ili N_{min}. Ako je x negativan, tada je je situacija analogna (zrcalna slika u odnosu na ishodište). IEEE standard definira *korektno zaokruženu vrijednost* od x, koja se označava s round(x) na sljedeći način. Ako je x broj s pomičnom točkom, tada je round$(x) = x$. Ako nije onda vrijednost od round(x) ovisi o načinu (modu) zaokruživanja koji je aktivan. Postoje četiri načina

- Zaokruživanje prema dolje (još se kaže prema $-\infty$): round$(x) = x_-$.
- Zaokruživanje prema gore (još se kaže prema ∞): round$(x) = x_+$.
- Zaokruživanje prema nuli: $\text{round}(x) = \begin{cases} x_- & \text{ako je } x > 0 \\ x_+ & \text{ako je } x < 0 \end{cases}$.
- Zaokruživanje prema najbližem:

$$\text{round}(x) = \begin{cases} x_- & \text{ako je } \mid x - x_- \mid < \mid x - x_+ \mid \\ x_+ & \text{ako je } \mid x - x_- \mid > \mid x - x_+ \mid \end{cases}.$$

  Ako je $\mid x-x_- \mid = \mid x-x_+ \mid$ uzima se x_- ili x_+, već prema tome je li u x_- ili u x_+ najmanje značajni bit b_{p-1} nula. Ako je $x > N_{max}$ uzima se round$(x) = +\infty$, a ako je $x < N_{min}$ uzima se round$(x) = -\infty$.

U praksi se gotovo uvijek koristi zadnji način zaokruživanja, prema najbližem susjedu u sustavu brojeva s pomičnom točkom i njega ćemo još malo pojasniti. Neka je x opet kao u (2.3.1). Ako je prvi bit koji se ne može reprezentirati, b_p jednak 0, tada je round$(x) = x_-$. Ako je $b_p = 1$ i još barem jedan od sljedećih bitova nije nula, tada je round$(x) = x_+$. Ako je $b_p = 1$ i svi preostali bitovi od x su 0, tada se uzima x_- ili x_+ ovisno o tome koji od njih ima najmanje značajan bit 0. Uočimo da se x_- i x_+ razlikuju samo u zadnjem bitu, pa će jedan sigurno biti izabran. IEEE standard zahtijeva da se kao prvi (uobičajen, default) način uzima zaokruživanje prema najbližem, pa će se nadalje, ako se ne spomene drugačije, round(x) koristiti

u tom smislu. Ako je $x > N_{max}$ i način zaokruživanja je prema najbližem, tada je $\text{round}(x) = \infty$ iako je x naravno bliže N_{max} nego ∞.

Zadatak 2.3.3. *Napišite u jednostrukom IEEE formatu reprezentacije zaokruženih vrijednosti (u načinu do najbližeg) brojeva:* $1/10$, $2 \cdot 2^{24}$ *i* 2^{131}. ■

Zadatak 2.3.4. *Konstruirajte x za koji su x_- i x_+ jednako udaljeni od x i nađite pripadnu reprezentaciju od x.* ■

Zadatak 2.3.5. *Neka x, $0 < x < N_{min}$ nije subnormalni broj. Tada je*

$$x = (0.b_1 b_2 \ldots b_{p-1} b_p b_{p+1} \ldots)_2 \times 2^{e_{min}},$$

pri čemu je bar jedan od brojeva $b_p b_{p+1} \ldots$ jednak 1. Što je x_-? Uzmi kao primjere brojeve 2^{-130} i 2^{-150}. (Pretpostavka je jednostruki format i $e_{min} = -126$). ■

Ako $x = (1.b_1 \ldots b_{p-1} b_p b_{p+1} \ldots)_2 \times 2^e$ nije reprezentabilan jasno je da je $x_- \leq x \leq x_+$ bez obzira na način zaokruživanja. Stoga je

$$| \text{round}(x) - x | < 2^{-(p-1)} \times 2^e$$

pa kažemo da je apsolutna greška kod zaokruživanja manja od jednog ulpa. Pritom se misli na $\text{ulp}(x_-)$ ako je $x > 0$ i $\text{ulp}(x_+)$ ako je $x < 0$. Kad je prisutno zaokružiavanje do najbližeg, onda vrijedi jača ocjena

$$| \text{round}(x) - x | \leq 2^{-p} \times 2^e$$

pa kažemo da je apsolutna greška zaokruživanja pola ulpa.

Zadatak 2.3.6. *Odredite apsolutnu grešku od* $1/10$ *ako se koristi jednostruki IEEE format. Koristite svaki od četiri načina zaokruživanja.* ■

Zadatak 2.3.7. *Vrijede li gornje ocjene za apsolutnu grešku zaokruživanja ako je $| x | < N_{min}$ Objasnite.* ■

Zadatak 2.3.8. *Kolika je apsolutna greška za svaki od načina zaokruživanja ako je $| x | < N_{min}$. Pazite na definiciju round(x).* ■

S obzirom da apsolutna greška kod zaokruživanja raste kad $| x |$ raste promotrimo relativnu grešku od $\text{round}(x)$ kao aproksimacije od x. Iz relacije (2.3.1) znamo da je $| x | \geq 2^e$. Stoga za sve načine zaokruživanja vrijedi

$$\frac{| \text{round}(x) - x |}{| x |} < \frac{2^{-(p-1)} \times 2^e}{2^e} = 2^{-(p-1)} \equiv \epsilon.$$

Kod zaokruživanja do najbližeg vrijedi malo bolja ocjena

$$\frac{\mid \mathrm{round}(x) - x \mid}{\mid x \mid} \leq \frac{2^{-p} \times 2^e}{2^e} = 2^{-p} \equiv \epsilon \,.$$

Koristeći oznake iz relacija (2.1.2) i (2.1.1) možemo za realne brojeve koji leže u normaliziranom intervalu napisati

$$\mathrm{round}(x) = x(1+\delta)\,, \quad \mid \delta \mid \leq \epsilon\,, \tag{2.3.2}$$

gdje je $\epsilon = 2^{-p}$ ako se koristi zaokruživanje do najbližeg i $\epsilon = 2^{-p+1}$ ako se koriste ostali načini zaokruživanja. Pritom je p preciznost računala. Uočimo da $\delta = \delta(x)$ ovisi o x i načinu zaokruživanja. S obzirom da računala u normalnom radu koriste zaokruživanje do najbližeg, ako neće biti drugačije naznačeno, $\epsilon = 2^{-p}$, pri čemu je $p = 24$ za jednostruki, $p = 53$ za dvostruki i $p = 64$ za prošireni format.

Zadatak 2.3.9. *Nađite x u normaliziranom intervalu BPT za koji je $\delta = 2^{-p}$ ako se koristi uobičajeni način zaokruživanja do najbližeg.* ■

Zadatak 2.3.10. *Da li vrijedi ocjena (2.3.2) za $\mid x \mid < N_{min}$?* ■

S obzirom da je $-\log_2(\delta(x)) > p$ (odnosno $p-1$ ako nije uobičajeni način zaokruživanja), vidimo da mjera $-\log_2(\delta(x))$ daje broj binarnih znamenaka do kojeg se x i $\mathrm{round}(x)$ podudaraju. Slično će $-\log_{10}(\delta(x)) > -\log_{10}(\epsilon)$ mjeriti broj decimalnih mjesta u kojima se x i $\mathrm{round}(x)$ podudaraju.

S obzirom da je

$$\frac{\mid \mathrm{round}(x) - x \mid}{\mid \mathrm{round}(x) \mid} < \frac{2^{-p+1} \times 2^e}{2^e} = 2^{-p+1} \equiv \epsilon\,,$$

odnosno, $|\mathrm{round}(x) - x|/|\mathrm{round}(x)| \leq 2^{-p} \times 2^e/2^e = 2^{-p} \equiv \epsilon$ ako je prisutno zaokruživanje do najbližeg, možemo pored relacije (2.3.2) koristiti i relaciju

$$\mathrm{round}(x) = \frac{x}{1+\delta}\,, \quad \mid \delta \mid \leq \epsilon\,. \tag{2.3.3}$$

Pritom δ iz relacije (2.3.3) ne mora biti jednak onom iz relacije (2.3.2). Relacije (2.3.2) i (2.3.3) su osnova za nalizu grešaka zaokruživanja koju ćemo upoznati kasnije.

2.3.4. Korektno zaokružene osnovne računske oprecije

Jedna od najznačajnijih značajki IEEE standarda je da zahtijeva vrlo poželjnu preciznost osnovnih računskih operacija: rezultat mora biti takav kao da je izračunat točno i tek onda zaokružen. Označimo sa $\oplus$, $\ominus$, $\otimes$ i $\oslash$ operacije $+$, $-$, $\times$, i $/$ kako su

stvarno implementirane u računalu. Ako sa $\circ$ i $\odot$ označimo odgovarajuće operacije u algebri i u računalu, tada IEEE standard zahtijeva da vrijedi

$$x \odot y = fl((x \circ y) = \text{round}(x \circ y) = ((x \circ y)(1+\delta)\,, \quad |\,\delta\,| \leq \epsilon\,, \tag{2.3.4}$$

gdje su $\circ \in \{+, -, \times, /\}$, $\odot \in \{\oplus, \ominus, \otimes, \oslash\}$, a x i y su reprezentabilni brojevi. Osim za četiri osnovne računske operacije, standard zahtijeva da (2.3.4) vrijedi i za unarnu operaciju drugog korijena. Konačnu aritmetiku koja zadovoljava (2.3.4) se kadkad naziva *korektno zaokruživajuća aritmetika.*

Lijeva strana u (2.3.4) je broj u skupu BPT, pa vrijedi: $1 \otimes x = x$, $x \oslash x = 1$, $0.5 \otimes x = x \oslash 2$ kao i važno svojstvo: ako su x i y brojevi u skupu BPT i vrijedi $x \ominus y = 0$, tada je $x = y$.

Koristeći izvod i samu relaciju (2.3.3), odmah vidimo da vrijedi i ocjena

$$x \odot y = fl(x \circ y) = \frac{x \circ y}{1+\delta}\,, \quad |\,\delta\,| \leq \epsilon\,, \tag{2.3.5}$$

pri čemu zadnji δ nije općenito jednak δ iz relacije (2.3.4).

Napomena 2.3.1. *U numeričkoj matematici se je uvriježila notacija $fl(x+y)$, $fl(x-y)$, $fl(x \times y)$, $fl(x/y)$ za operacije $x \oplus y$, $x \ominus y$, $x \otimes y$ i $x \oslash y$. Također, fl(izraz) označava izračunatu vrijednost izraza pomoću operacija $\oplus$, $\ominus$, $\otimes$ i $\oslash$. Slično, $fl(f(x))$ označava izračunatu vrijednost funkcije f u točki x.* ■

Zadatak 2.3.11. *Koji je najveći broj x iz skupa BPT za koji je $1 \oplus x = 1$. Pretpostavke su: IEEE jednostruki format i zaokruživanje do najbližeg. Uz iste uvjete, koliko je $1 \oplus round(10^{-5})$, $1 \oplus round(10^{-10})$, $1 \oplus round(10^{-15})$?* ■

Iz same definicije operacija $\oplus$, $\ominus$ i $\otimes$ slijedi $x \oplus y = y \oplus x$, $x \ominus y = -(y \ominus x)$ i $x \otimes y = y \otimes x$. Međutim, već za malo složenije izraze, ono što vrijedi u aritmetici, ne vrijedi u konačnoj algebri s IEEE standardom.

Primjer 2.3.1. *Neka je $x = 1$, $y = 2^{-25}$, $z = 1$. Ti brojevi su u skupu BPT. Kako je $y = 1.0 \times 2^{-25}$, suma $x + y = 1.00000000000000000000000001$ se ne može egzaktno reprezemtirati u IEEE jednostrukom formatu, pa će uz zaokruživanje prema najbližem biti $x \oplus y = 1$. Stoga će vrijediti $(x \oplus y) \ominus z = 1 \ominus z = 0$. S druge strane, egzaktni rezultat $(x+y) - z = 2^{-25}$ je reprezentabilan u sustavu brojeva BPT, pa vrijedi $round((x+y)-z) = 2^{-25}$. Pokazali smo da je $fl((x+y)-z) = 0$ iako je $round((x+y)-z) = 2^{-25}$, pa za taj izraz ne vrijedi pravilo koje je aksiom IEEEE standrda za osnovne računske operacije.* ■

Slično, asocijativnost zbrajanja i distributivnost množenja prema zbrajanju, neće u IEEE aritmetici uvijek vrijediti.

Zadatak 2.3.12. *Neka su* $x = 1$, $y = 2^{-25}$, $z = 1.1 \times 2^{-25}$. *Uz pretpostavku IEEE jednostrukog formata i zaokruživanja do najbližeg izračunajte* $a \equiv (x \oplus y) \oplus z$ *i* $b \equiv x \oplus (y \oplus z)$. *Pokažite da* a *i* b *nisu jednaki!*
Rješenje. $x \oplus y = 1$, $(x \oplus y) \oplus z = 1 \oplus z = 1$, $y \oplus z = 2.1 \times 2^{-25}$, $x \oplus (y \oplus z) = 1 \oplus 2.1^{-25} = 1 + 2^{-23}$. ■

2.3.5. Implementacija operacija na računalu

Počnimo od zbrajanja i oduzimanja, koje nećemo odvajati jer je $x - y = x + (-y)$. Ako operandi nemaju isti eksponent, onda se onaj (po modulu) manji operand napiše u obliku nenormaliziranog broja s eksponentom većeg. To znači da mu se mantisa pomakne u desno za onoliki broj binarnih mjesta kolika je na početku bila razlika u eksponentima operanada. Zatim se rezultat svede u normalizirani oblik, pri čemu se prvo mantisa pomakne u lijevo ili u desno ako je potrebno, zatim zaokruži i ako je nakon zaokruživanja potrebno, opet se pomakne. Svako pomicanje mantise rezultira istovremenim namještavanjem eksponenta. Dakle, postupak nije najjednostavniji.

Da bi se implementirala IEEE aritmetika potrebni su i tzv. *sigurnosni* (*dodatni, zaštitni, rezervni*) bitovi. To znači da aritmetika u računalu koristi registre koji sadrže riječ s više bitova nego što zahtijeva standard za reprezentaciju brojeva. Promotrimo računanje razlike $x - y$ u jednostrukom formatu, pri čemu je $x = (1.0)_2 \times 2^0$ i $y = (1.111\ldots1)_2 \times 2^{-1}$. Izjednačavanjem eksponenata, dobije se

$$\begin{aligned} & (1.00000000000000000000000|\ \ )_2 \times 2^0 \\ - & (0.11111111111111111111111|1\ )_2 \times 2^0 \\ = & (0.00000000000000000000000|1\ )_2 \times 2^0 \\ \text{normalizacija: } = & (1.00000000000000000000000|0\ )_2 \times 2^{-24} \end{aligned}$$

Ovdje smo koristili jedan dodatni bit (iza okomite crte | koja prikazuje granicu). Ovo je primjer *kraćenja* jer se skoro svi bitovi u dva broja pokrate.

Sljedeći primjer pokazuje da kadkad nije dovoljan jedan bit. Razmotrimo računanje $x - y$ u jednostrukom formatu uz zaokruživanje do najbližeg, pri čemu su $x = (1.0)_2 \times 2^0$ i $y = (1.000\ldots01)_2 \times 2^{-24}$ brojevi iz BSPT. Koristeći (čak) 25 dodatnih bitova, dobivamo

$$\begin{aligned} & (1.00000000000000000000000|\ \ )_2 \times 2^0 \\ - & (0.00000000000000000000000|01000000000000000000001\ )_2 \times 2^0 \\ = & (0.11111111111111111111111|10111111111111111111111\ )_2 \times 2^{-1} \\ & \qquad\qquad \text{normalizacija:} \\ = & (1.11111111111111111111111|01111111111111111111110\ )_2 \times 2^{-24} \\ & \qquad\qquad \text{zaokruživanje:} \\ = & (1.11111111111111111111111|\ \ )_2 \times 2^{-24} \end{aligned}$$

Da smo koristili samo dva dodatna bita rezultat bi (nakon dodatne renormalizacije) bio $(1.00000000000000000000000)_2 \times 2^0$. Isti krivi rezultat dobivamo korištenjem 3,4, ..., 24 dodatnih bitova. Na sreću, umjesto 25 (ili u sličnim namještenim slučajevima po volji veliki broj) dodatnih bitova, dovoljno je koristiti tek tri dodatna bita, točnije, dva zaštitna i jedan *zaljepljeni* bit. Zovemo ga zaljepljeni jer kad ga aktiviramo on se više ne mijenja. On se aktivira onda kad je potrebno pomaknuti mantisu za više od dva bita i ta jedinica se stavlja iza drugog zaštitnog bita. Dakle, u zadnjem primjeru, prije oduzimanja treba dodati zaljepljenu jedinicu iza drugog zaštitnog bita:

$$\begin{aligned}
&(1.00000000000000000000000| \quad)_2 \times 2^0 \\
-\ &(0.00000000000000000000000|011\ )_2 \times 2^0 \\
=\ &(0.11111111111111111111111|101\ )_2 \times 2^{-1} \\
\text{normalizacija:} =\ &(1.11111111111111111111111|01\ \ )_2 \times 2^{-24} \\
\text{zaokruživanje:} =\ &(1.11111111111111111111111| \quad)_2 \times 2^{-24}
\end{aligned}$$

Množenje i dijeljenje u BSPT ne zahtijeva poravnavanje eksponenata. Ako su $x = m_x \times 2^{e_x}$ i $y = m_y \times 2^{e_y}$, tada je $z = x \cdot y = (m_x \cdot m_y) \times 2^{e_x+e_y}$, pa kod množenja u BSPT postoje tri koraka: pomnoži mantise, zbroji eksponente, te ako je potrebno normaliziraj produkt mantisa i pravilno zaokruži rezultat. Kod dijeljenja ćemo isto tako imati kvocijent mantisa, razliku eksponenata i normalizaciju sa zaokružiavanjem. Današnji dizajneri chipova su uspjeli, uz dovoljnu rezervu memorije toliko ubrzati množenje da je gotovo jednako brzo kao i zbrajanje odnosno oduzimanje. Jedino treba voditi računa da je dijeljenje još uvijek nekoliko puta sporije od množenja. Dijeljenje s nulom ćemo razmotriti kasnije.

Uočimo da $1 \leq m_x < 2$ i $1 \leq m_y < 2$ povlače $1 \leq m_z = m_x \cdot m_y < 4$. Dakle, lijevo od binarne točke u binarnoj reprezentaciji od m_z mogu biti samo kombinacije bitova `11`, `10` ili `01` tj. `1`. Stoga se mogući izbor pomaka mantise svodi najviše za jedno mjesto u lijevo uz istodobno povećanje eksponenta za 1.

Kod dijeljenja će vrijediti $1/2 < m_x/m_y < 2$, pa će pomicanje kod normalizacije biti najviše jedan bit u lijevo ili u desno.

Zadatak 2.3.13. *Odgovorite na sljedeća pitanja za slučaj:*
(a) brojevnog sustava za prikazivanje ($p = 3$, $\epsilon_M = 0.25$, $-1 \leq e \leq 1$) i
(b) BSPT u jednostrukom IEEE formatu.

(i) Koliko brojeva x s pomičnom točkom zadovoljava $1 \leq x < 2$? Koliko od njih zadovoljava $1 \leq x < 3/2$, a koliko $3/2 \leq x < 2$?
(ii) Koliko brojeva y s pomičnom točkom zadovoljava $1/2 < y \leq 1$? Koliko od njih aproksimativno zadovoljava $1/2 < x \leq 2/3$, a koliko $2/3 < x \leq 1$?
(iii) Slijedi li da moraju postojati dva različita broja s pomičnom točkom između 1 i 2 za koje su izračunate recipročne vrijednosti $1 \oslash x_1$ i $1 \oslash x_2$ jednake (zaokružene u istom formatu)? Mislite li na x_1 i x_2 između 1 i 3/2 ili između 3/2 i 2? Vrijedi li to bez obzira na način zaokruživanja?

(iv) Slijedi li da postoje brojevi s pomičnom točkom za koje $(1 \oslash x) \otimes x$ *nije jednako* 1. *Vrijedi li to za svaki način zaokruživanja? Da li također slijedi da postoje brojevi s pomičnom točkom za koje* $1 \oslash (1 \oslash x)$ *nije jednako* x. ■

2.3.6. Drugi korijen, ostatak pri dijeljenju i konverzija formata

Pored zahtjeva da su osnovne aritmetičke operacije korektno zaokružene, IEEE standard zahtijeva da su korektno zaokružene i operacija drugog korijena i ostatak pri dijeljenju. Drugi korijen je unarna operacija (jer je funkcija) definirana za nenegativni operand pri čemu u implementaciji (koja je najčešće hardwareska) vrijedi $\sqrt{-0} = -0$. Ostatak pri dijeljenju realnih brojeva x i y je realni broj $x\ REM\ y = x - y \times n$, gdje je n cijeli broj najbliži kvocijentu x/y.

Brojevi ulaze u računalo na nekoliko načina. Kod računanja na računalu najčešće se koriste viši programski jezici koji procesiraju naredbe kao kompileri ili interpreteri. Ako npr. broj 0.4 sudjeluje u računu njega možemo u programu uvesti kao konstantu ili pridružiti tu vrijednost nekoj varijabli. Pridruživanje varijabli može biti u naredbi oblika $x = 0.4$, a može x dobiti vrijednost učitavanjem broja 4 sa neke vanjske memorije, npr. iz neke tekst datoteke na disku. Kompiler ili interpreter tada poziva standardne rutine odnosno potprograme za ulaz podataka. Te rutine generiraju strojne instrukcije koje prevode decimalni broj, tj. niz znakova (koji su: decimalne znamenke, točka i možda slova kao e, E, d, D koje označavaju početak eksponencijalnog dijela broja) u binarni format i smjeste ga kao korektno zaokruženi broj u varijablu memorije ili registra. Isto tako je moguće razlomak 4/10 unijeti kao cijele ili realne brojeve 4 i 10 i unutar programa koristiti operaciju dijeljenja da bi izračunali 0.4. Ako se 4 i 10 unašaju kao cijeli (realni) brojevi, kompiler ili interpreter će generirati strojne instrukcije koje će nizove znakova '4' i '10' pretvoriti u njihove cjelobrojne (ili realne IEEE) binarne reprezentacije. U svakom slučaju prije nego će se pozvati komanda za dijeljenje, operandi moraju biti u binarnoj reprezentaciji pomične točke jer će rezultat biti u tom formatu. Kod izlaza podataka iz računala, npr. kod ispisa ili spremanja rezultata u tekst datoteku, kompiler ili interpreter (kraće kažemo – računalo) će generirati instrukcije za konverziju binarnih formata u decimalne brojeve, koje smo navikli prepoznati.

IEEE standard zahtijeva podršku za korektno prevođenje između raznih formata. Tu spadaju sljedeće pretvorbe ili konverzije.

Konverzije između formata brojeva s pomičnom točkom. Konverzija iz kraćeg u duži format (npr. iz jednostrukog u dvostruki) mora biri egzaktna. Konverzije iz dužeg u kraći zahtijeva zaokruživanje.

Konverzija iz BSPT u cjelobrojni format zahtijeva zaokruživanje do najbližeg cijelog broja pomoću načina zaokruživanja koji je aktivan. Ako je broj u BSPT cijeli,

konverzija mora dati isti cijeli broj, osim ako se taj cijeli broj ne može realizirati u danom cjelobrojnom formatu. Obrnuta konverzija, iz cjelobrojnog formata u format s plivajućom točkom će možda zahtijevati zaokruživanje. Također se zahtijeva zaokruživanje broja u iz skupa BSPT u cijeli broj reprezentiran istim formatom.

Konverzije iz decimalnog u binarni sustav i obrnuto. Pritom se koristi način zaokruživanja koji je aktivan. Nakon 1985. kad je uveden IEEE 754 standard, pronađeni su novi i/ili efikasniji algoritmi za korektnu konverziju između svih tipova formata i oni imaju često bolja svojstva nego što standard zahtijeva.

2.3.7. Izuzeci

U matematici su realni ili cijeli brojevi uvijek konačni. Oznake ∞ ili $-\infty$ tek služe za naznačavanje da vrijednost neke veličine (npr. opći član niza brojeva) teži prema sve većim ili manjim brojevima. U ovom dijelu teksta, misleći na brojeve s kojima računalo radi, spominjemo ∞ i $-\infty$ kao brojeve. Isto tako $+0$ i -0, jer imaju različitu strojnu reprezentaciju, spominjemo kao dva različita broja, iako imaju istu vrijednst 0.

Najjednostavniji primjer jedne izuzetne situacije je dijeljenje s nulom. Prije IEEE standarda su neki proizvođači računala, rezultat dijeljenja pozitivnog broja s nulom definirali najvećim prikazivim brojem. Oni su to obrazlagali tvrdnjom da kada korisnik primijeti vrlo velike brojeve, znat će da je nešto krenulo loše. Tada bi rezultat operacije kao što je $1/0 - 1/0$ bio 0, pa to korisnik vjerojatno ne bi primijetio, pogotovo ako se takva stvar dogodila usred složenijg računanja. Stoga se nakon 1960. među proizvođačima računala usvojilo pravilo da se izvršavanje programa prekida ako se broj dijeli s nulom, uz poruku oblika: "fatalna greška – dijeljenje s nulom". Na taj način je bilo otežano programiranje, jer se smatralo da je odgovornost programera da do djeljenja s nulom ne dođe. Kako je to razriješio IEEE standard?

Matematički je jasno da vrijedi: $a \times 0 = 0$ za svaki konačni broj a, $a/0 = \infty$ za svaki pozitivni broj a, $a \times \infty = \infty$ za svaki pozitivni a. S druge strane izrazi poput $0 \times \infty$, $0/0$, ∞/∞ nemaju smisla, pa je najbolje da računalo to tretira kao nepravilnu operaciju. Stoga IEEE standard zahtijeva da se svakoj takvoj operaciji pridruži vrijednost NaN (Not a Number - nije broj). Ako se NaN pojavi u nekom izrazu, tada cijeli izraz dobiva vrijednost NaN. Na kraju će možda sve varijable s izlaznim vrijednostima imati sadržaj NaN. Na taj način će doći poruka programeru da je nešto krenulo loše.

Zbrajanja u kojima se pojavljuje ∞ uglavnom imaju smilsla: $a + \infty = \infty$, $a - \infty = -\infty$ za svaki konačni broj a, pa je i $\infty + \infty = \infty$, $-\infty + (-\infty) = -\infty$. To se lako pokaže pomoću konvergirajućih (prema a) i divergirajućih (prema ∞) nizova. Međutim, $\infty - \infty$ nema smisla pa je $\infty - \infty = NaN$. Slično, $\infty/0 = \infty$,

$0/\infty = 0$, dok $0/0$ i ∞/∞ nemaju smisla jer mogu u interpretaciji nizova biti bilo koji brojevi. Dakle je $0/0 = NaN$ i $\infty/\infty = NaN$.

S obzirom da standard uvodi predznake ispred posebnih brojeva 0 i ∞: $+0$ je isto što i 0 pa koristimo 0, -0; $+\infty$ je isto što i ∞ pa koristimo ∞ i $-\infty$, pogledajmo kako ih iskoristiti. Ako je a pozitivni konačni broj, stavljamo $a/0 = \infty$, $a/-0 = -\infty$, $-a/-0 = \infty$, $-a/0 = -\infty$ kao i $a/\infty = 0$, $a/-\infty = -0$, $-a/\infty = -0$, $-a/-\infty = 0$. Predikat $0 = -0$ je istinit, dok je predikat $\infty = -\infty$ lažan. Na taj način smo došli do toga da implikacija $a = b \Rightarrow 1/a = 1/b$ ne vrijedi ako je $a = 0$ i $b = -0$.

Upotreba NaN-ova je prikladna ako se zahtijeva vrijednost funkcije za argument koji nije u domeni funkcije, npr. rezultat operacije $\sqrt{-3.2}$ će biti NaN. Također je zgodno, polazne vrijednosti varijabli koje nemaju polazne vrijednosti definirati s NaN, jer će nam to otkriti grešku njihova preranog korištenja (prije nego dobiju neku vrijednost) u programu. NaN nije u uređaju ni sa jednim brojem, a rezultat operacije $a \odot NaN$ je NaN za svaki reprezentabilni broj i svaku aritmetičku operaciju $\odot$.

Zadatak 2.3.14. *Ima li još slučajeva kad implikacija $a = b \Rightarrow 1/a = 1/b$ ne vrijedi. Koje su vrijednosti izraza: $0/-0$, $\infty/-\infty$ i $-\infty/-0$?* ■

2.3.8. Prekoračenje, potkoračenje i postepeno potkoračenje

O *prekoračenju* (engl. overflow) se govori kada je egzaktni rezultat neke operacije konačan broj, ali po modulu je veći od najvećeg prikazivog broja u BSPT kojeg označavamo s N_{max}. Uočimo da N_{max} ovisi o binarnom formatu. Prema IEEE standardu takav broj treba zaokružiti prema načinu zaokruživanja koji se koristi. Ako je broj koji prekoračuje pozitivan, tada će zaokruživanje prema gore zaokružiti broj na ∞, dok će zaokruživanje prema dolje i prema nuli broj zaokružiti na N_{max}. Zaokruživanje prema najbližem, odudara od ove matematičke logike jer će broj zaokružiti na ∞. Razlog je praktične naravi, radi se o osnovnom modu zaokruživanja koji se gotovo uvijek koristi, a kad bi se broj zaokružio na (matematički najbliži BPT) N_{max} računanje bi moglo dati rezultat koji zavarava – izgleda kao da je sve u redu, a nije.

Potkoračenje (engl. underflow) nastaje kada je egzaktni rezultat neke operacije od nule različiti broj koji je po modulu manji od od najmanjeg pozitivnog prikazivog broja u BSPT kojeg označavamo s N_{min}. Prije IEEE standarda tipični odgovor računala je bio stavljanje broja-rezultata na nulu. U IEEE aritmetici, standardni odgovor je pravilno zaokruženi broj (prema aktivnom načinu u BSPT) koji je možda subnormalni broj. Taj događaj se zove *postepeno potkoračenje* (engl. gradual underflow) i on je i danas najkontroverzniji dio IEEE standarda. Najme, postepeno potkoračenje daje rezultate koji imaju manju (to manju što je potkoračenje veće)

relativnu točnost. Međutim, i to je u većini slučajeva bolje nego definirati rezultat nulom i tako potpuno izgubiti relativnu točnost. Osim toga postepeno potkoraćenje osigurava lijepo svojstvo da za brojeve u BSPT vrijedi implikacija $x \ominus y = 0 \Rightarrow x = y$.

Primjer 2.3.2. *Neka je* $y = 0.1 \times 2^{-123}$, $x = 25 \times 2^{-131}$ *i* $z = y \ominus x$. *Uočimo da je* $25 = (11001)_2$. *Računaje od z u jednostrukom IEEE formatu daje*

$$\begin{aligned} & (1.10011001100110011001101\ )_2 \times 2^{-127} \\ - & (1.10010000000000000000000\ )_2 \times 2^{-127} \\ = & (0.00001001100110011001101\ )_2 \times 2^{-127} \\ \textit{normalizacija:} = & (1.10011001100110011010000\ )_2 \times 2^{-132} \end{aligned}$$

Pritom je zadnja jedinica u binarnom prikazu od x nastala zbog zaokruživanja. Vidimo da je rezultat subnormalni broj, koji je što se relativne preciznosti tiče izgubio zadnja četiri bita što odgovara približno jednoj decimalnoj znamenki. Bez postepenog potkoraćenja, rezultat bi bio nula. ■

Zadatak 2.3.15. *Promotrimo operaciju* $(y \ominus x) \oplus x$, *pri čemu rezultat prve operacije potkoračuje. Koji je rezultat ako postoji i ako ne postoji postepeno potkoračenje? Koristite prethodni primjer.* ■

Zadatak 2.3.16. *Neka su x i y brojevi u BSPT za koje vrijedi*

$$\frac{1}{2} \leq \frac{x}{y} \leq 2\,.$$

Pokažite da je tada $x - y$ *broj u BSPT, tj. da je tada oduzimanje egzaktno.* ■

IEEE standard ukupno definira pet vrsta izuzetnih situacija koje su ukratko opisane u sljedećoj tabeli.

pogrešna operacija	rezutat je NaN
dijeljenje s nulom	rezultat je $\pm\infty$
prekoračenje	rezultat je $\pm\infty$ ili $\pm N_{max}$
potkoračenje	rezultat je ± 0 ili $\pm N_{min}$ ili subnormalni broj
ne egzaktni rezultat	rezultat je korektno zaokruženi broj

Tablica 2.3.5. IEEE odgovor na izuzetne situacije

IEEE standard specificira da se svaki izuzetni događaj mora signalizirati pomoću tzv. statusnog signala (engl. status flag) kojeg bi programer mogao dohvatiti tj. iskoristiti za pravljenje programa (vidi sljedeći primjer) ili pustiti računalo da daje standardne ispise kao u zadnjoj tabeli, za vrijeme izvršavanja programa.

Primjer 2.3.3. *Računanje izraza $\sqrt{x^2+y^2}$ daje jednostavan primjer kako se korištenjem IEEE standarda po pitanju izuzetaka mogu načiniti brzi i pouzdani programi. Pravilo je: pokušaj prvo na jednostavan i brz način, a ako dođe do izuzetne situacije, računaj ponovo na siguran, ali spor način. U ovom primjeru, ako je $\mid x \mid$ ili $\mid y \mid$ veći od $\sqrt{N_{max}}$, doći će do prekoračenja. Tada se $z = \sqrt{x^2+y^2}$ računa pomoću algoritma:*

$u := \max(\lvert x\rvert, \lvert y\rvert)$;	*Ovaj algoritam zahtijeva kao i onaj direktni dva množenja, jedno zbrajanje i jedan korijen, međutim dodatno zahtijeva izvršavanje nekoliko funkcija:* $\lvert\cdot\rvert$, max *i* min, *pa je nekoliko puta sporiji i nešto netočnij. (vidi primjer 2.5.1.).*
$v := \min(\lvert x\rvert, \lvert y\rvert)$;	
if $(u = 0)$ $z := 0$	
else $z := u\sqrt{1+(v/u)^2}$	
endif	

Sa IEEE aritmetikom, prvo se izračuna $z = \sqrt{x^2+y^2}$ direktno. Ako se pojavi izuzetna situacije sa prekoračenjem (što će biti vrlo rijedak slučaj), tek tada treba uključiti sofisticiraniji algoritam. ■

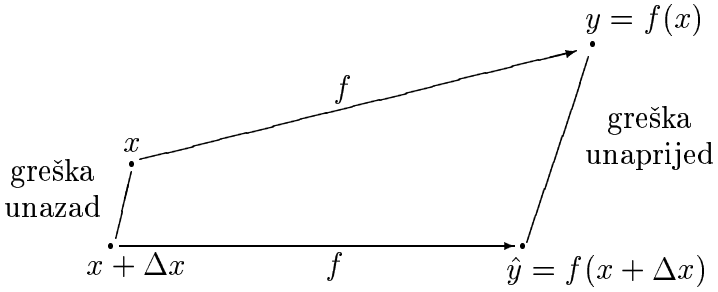
2.4. Stabilnost numeričkog računanja

U ovom dijelu ćemo se baviti stabilnošću numeričkih algoritama, pojmom koji se tiče pozdanosti dobivenih rješenja. Susrest ćemo niz pojmova koji određuju ili opisuju točnost s kojom algoritam računa izlazne podatke, u prisustvu grešaka zaokruživanja koje su neizbježne kad se koristi konačna aritmetika. Kroz primjere upoznat ćemo neke neugodne fenomene koji se mogu pojaviti kod korištenja konačne odnosno strojne aritmetike. Većina primjera je uzeta iz knjige [2].

2.4.1. Greške unazad i unaprijed

Neka je f realna funkcija realne varijable. Pretpostavimo da se u aritmetici preciznosti ϵ vrijednost $y = f(x)$ izračuna kao $\hat{y}$. Kako možemo mjeriti kvalitetu $\hat{y}$ kao aproksimacije od y?

U većini slučajeva bit ćemo sretni ako postignemo malu relativnu grešku u rezultatu, tj. $E_{rel} \approx \varepsilon$, ali to se neće moći uvijek postići. Umjesto toga možemo se zapitati *za koji skup podataka smo zapravo riješili problem?*. Dakle, za koji Δx imamo $\hat{y} = f(x + \Delta x)$. Općenito će biti više takovih Δx pa će nas zanimati najmanji. Vrijednost $|\Delta x|$ (ili $\min|\Delta x|$) možda podijeljena sa $|x|$ se zove greška unazad (engl. backword error) ili povratna greška. Apsolutna i relativna greška od $\hat{y}$ se zovu greške unaprijed ili kraće greške. Proces omeđivanja (tj. traženja ograda za) povratne greške izračunatog rješenja zove se analiza povratne greške ili povratna analiza greške (backword error analysis), a motivacija za taj postupak je dvostruka.



Slika 2.4.1. Greške unaprijed i unazad

Prvo, ona interpretira greške zaokruživanja kao greške u podacima. Podaci često kriju netočnosti od prethodnih izračunavanja, zbog spremanja u računalo ili jer su rezultat mjerenja. Ako povratna greška nije veća od tih polaznih netočnosti, tada se izračunato rješenje ne može mnogo kritizirati jer je to rješenje koje tražimo do na "ulaznu" netočnost. Druga privlačnost povratne analize grešaka je što ona reducira problem omeđivanja greške unaprijed na primjenu teorije perturbacije za dani problem. Teorija perturbacije je dobro poznata za većinu problema i važno je da ona ovisi o problemu, a ne o pojedinoj metodi za dani problem. Kada dobijemo ocjenu za povratnu grešku rješenja kod primjene određene metode, tada dodatnom primjenom opće teorije perturbacije za dani problem lako dolazimo do ocjene za grešku unaprijed.

Metoda za računanje vrijednosti $y = f(x)$ je povratno stabilna ili stabilna unazad (backward stable), ako ona za svako x producira izračunati $\hat{y}$ sa malom povratnom greškom, tj. vrijedi $\hat{y} = f(x + \Delta x)$ za malo Δx. Oznaka *malo* ovisi o kontekstu. U načelu: za dani problem može postojati više metoda od kojih će neke biti povratno stabilne, a neke neće biti.

Npr. sve osnovne računske operacije u računalu zadovoljavaju relaciju (2.1.1) pa daju rezultat koji je točan za malo pomaknute polazne podatke: $x \to x(1+\delta)$ i $y \to y(1+\delta)$ uz $|\delta| \leq \epsilon$. Dakle, sve su operacije u računalu povratno stabilne.

Međutim, većina metoda za računanje funkcije $\cos(x)$ ne zadovoljava relaciju $\hat{y} = \cos(x + \Delta x)$ za malo Δx, već samo slabiji rezultat: $\hat{y} + \Delta y = \cos(x + \Delta x)$ uz male Δx i Δy. Rezultat pisan u obliku

$$\hat{y} + \Delta y = f(x + \Delta x), \quad |\Delta y| \leq \eta |y|, \quad |\Delta x| \leq \xi |x| \tag{2.4.1}$$

se naziva miješani naprijed-nazad rezultat. Ako su ξ i η u (2.4.1) mali, može se reći: izračunato rješenje $\hat{y}$ se jedva razlikuje od vrijednosti $\hat{y} + \Delta y$ koje se dobije **egzaktnim računom** na ulaznoj vrijednosti $x + \Delta x$ koja se jedva razlikuje od stvarnog ulaznog podatka x.

Algoritam je numerički stabilan ako je stabilan u smislu relacije (2.4.1) s malim ξ i η. Ova definicija se uglavnom odnosi na izračunavanja u kojima su greške zaokruživanja (osnovnih aritmetičkih operacija) dominantni oblici grešaka. Inače pojam stabilnosti ima različita značenja u drugim područjima numeričke matematike.

2.4.2. Uvjetovanost

Odnos između greške unaprijed i greške unazad za dani problem u velikoj mjeri je određen uvjetovanošću problema, tj. osjetljivošću rješenja problema na ulazne podatke.

Pretpostavimo da je dano približno rješenje $\hat{y}$ problema $y = f(x)$ koje zadovoljava $\hat{y} = f(x + \Delta x)$. Ako pretpostavimo da je f dvaput neprekidno derivabilna, razvoj u Taylorov red daje

$$\hat{y} - y = f(x + \Delta x) - f(x) = f'(x)\Delta x + \frac{f''(x + \Theta\Delta x)}{2!}(\Delta x)^2, \quad \Theta \in (0, 1)$$

i možemo ocijeniti desnu stranu. Jer je

$$\frac{\hat{y} - y}{y} = \frac{f'(x)}{f(x)}\Delta x + \frac{f''(x + \Theta\Delta x)}{2f(x)}(\Delta x)^2,$$

imamo

$$\frac{\hat{y} - y}{y} = \frac{xf'(x)}{f(x)}\frac{\Delta x}{x} + \mathcal{O}\left((\Delta x)^2\right),$$

pa veličina

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

mjeri relativnu promjenu od y za malu relativnu promjenu od x. Zato $c(x)$ možemo zvati (relativni) broj uvjetovanosti od f, ili kraće uvjetovanost od f. Ako su f ili x vektori, tada se broj uvjetovanosti definira na slični način koristeći normu. Uvjetovanost služi za mjerenje najveće relativne promjene koja se dostiže za neku vrijednost broja ili vektora x. Npr. za $f(x) = \ln(x)$ imamo $c(x) = 1/\mid \ln(x) \mid$, pa je uvjetovanost velika za $x \approx 1$. To znači da mala relativna promjena u x uvijek izazove malu apsolutnu promjenu promjenu u $f(x) = \ln(x)$ (jer je $f(x+\Delta x)-f(x) \approx f'(x)\Delta x = \Delta x/x$), ali i veliku relativnu promjenu za neke x.

Kad se greška unaprijed, greška unazad i uvjetovanost, za dani problem definiraju na konzistentni način, vrijedi jednostavno pravilo:

$$\text{greška unaprijed} \lessapprox \text{uvjetovanost} \times \text{greška unazad}.$$

Dakle, izračunato rješenje loše uvjetovanog problema može imati veliku grešku unaprijed. Čak i kad izračunato rješenje ima malu grešku unazad, prelazom na grešku unaprijed, ona se može povećati za faktor velik kao broj uvjetovanosti. Zato se uvodi sljedeća definicija:

Ako metoda daje rješenja s greškama unaprijed koja su sličnog reda veličine kao ona koja se dobiju primjenom povratno stabilne metode, tada se za metodu kaže da je stabilna unaprijed.

Dakle, sama metoda ne treba biti povatno stabilna da bi bila stabilna unaprijed. Povratna stabilnost implicira stabilnost unaprijed, dok obrat ne vrijedi (primjer je npr. Cramerovo pravilo za 2×2 linearne sustave).

2.4.3. Akumulacija grešaka zaokruživanja

Dosta je rasprostranjeno mišljenje da velike brzine modernih računala koje omogućavaju u svakoj sekundi izvršavanje milijarde računskih operacija imaju pri zahtjevnim proračunima za posljedicu potencijalno zastrašujući velike greške u rezultatu. Na sreću ta tvrdnja uglavnom nije istinita, a u rijetkim slučajevima kada dolazi do većih grešaka u rezultatu, kriva je jedna ili tek nekoliko podmuklih grešaka zaokruživanja.

Primjer 2.4.1. *Slučaj složenih kamata. Ako se a novčanih jedinica investira na godinu dana po godišnjoj kamatnoj stopi x (izraženoj u decimalnom obliku, npr. $x = 0.04$) uz n ukamaćivanja (npr. za kvartalno ukamaćivanje je $n = 4$), tada je buduća vrijednost uloženog novca nakon godine dana formulom*

$$C_n(x) = a\left(1 + \frac{x}{n}\right)^n.$$

To je tzv. formula složenog ukamaćivanja. Poznato je da kad broj ukamaćivanja tijekom godine n raste, $C_n(x)$ monotono raste prema vrijednosti e^x. U graničnom slučaju, kad $n \to \infty$ govorimo o neprekidnom ukamaćivanju. Taj slučaj se u praksi manji susreće u bankarstvu, a više u biološkim i medicinskim područjima. Tada se govori o prirastu šume, razmnožavanju kunića u nekoj populaciji ili širenju zaraze u medicini. Nas će ovdje zanimati ulaganje novca pa ćemo pretpostaviti da je n konačan. Proučit ćemo stabilnost nekoliko algoritama za računanje $C_n(x)$. No prvo ćemo naći uvjetovanost od $C_n(x)$. Imamo

$$\kappa(C_n(x)) = \left|\frac{xC_n'(x)}{C_n(x)}\right| = \left|\frac{x}{1 + \frac{x}{n}}\right| = \frac{|\, x \,|}{|\, 1 + \frac{x}{n} \,|},$$

pa uvjetovanost konvergira prema x (provjerite da je to uvjetovanost od e^x) kad $n \to \infty$. Dakle je formula za uvjetovanost male kondicije i za veliko n, osim kad je x veliko. Promotrimo sljedeće algoritme za računanje $C_n(x)$.

Algoritam 2.4.1.

```
z:=1.0+x/n;  w:=1.0;
for i:=1 to n w:=w*z;
C:=w;
```

Algoritam 2.4.2.

```
z:=1.0+x/n;
C:=pow(z,n);
```

Kako n može biti veliki, pametno je iskoristiti operaciju (ili funkciju) potencije ako postoji. U FORTRANU se x^n piše $x**n$, dok se u jeziku C (slično je u Pascalu) poziva funkcija pow(x,n). U Matlabu je oblika `x^n` *ili* `x^(n)`*. Prednost je u tome što algoritam koji stoji iza operacije ili funkcije potencije zahtijeva oko* $\log_2(n)$ *množenja umjesto n množenja kao u prethodnom algoritmu (npr. x^{17} se računa kao $x \cdot ((((x^2)^2)^2)^2)^2$, pa koristi 5 množenja). Što je manje množenja, bit će i manje grešaka zaokruživanja.*

Sljedeća je mogućnost iskoristiti identitet $z^n = e^{n\ln(z)}$. Pritom možemo koristiti sljedeća dva algoritma

Algoritam 2.4.3.

```
z:=1.0+x/n;
w:=log(z);
C:=exp(n\times w);
```

Algoritam 2.4.4.

```
C:=exp(n\times log(1.0+x/n);
```

U tabeli su prikazani rezultati koje daju sva četiri algoritma. Programi su napisani u jeziku FORTRAN 77 i korišten je Digital Visual Fortran 6.0 compiler za PC računala. Korišten je (REAL) format, ali da bi dobili referentne (gotovo točne) podatke korišten je i potprogram za računanje složenih kamata u dvostrukom formatu (DOUBLE PRECISION). U sljedećoj tabeli dani su izlazni rezultati

n	Algor. 2.4.1.	Algor. 2.4.2.	Algor. 2.4.3.	Algor. 2.4.4.
4	1.050946	1.050946	1.050946	1.050945
12	1.051163	1.051163	1.051163	1.051162
365	1.051262	1.051262	1.051262	1.051268
1000	1.051215	1.051216	1.051216	1.051270
10000	1.051331	1.051342	1.051342	1.051271
100000	1.047684	1.048839	1.048839	1.051271
1000000	1.000000	1.000000	1.000000	1.051271

Zadnji stupac daje točne rezultate u danom formatu. To je zato jer zadnji algoritam koristi naredbu `C:=exp(n×log(1.0+x/n)` *koja se zapravo izvršava u registrima aritmetičke jedinice. Kako su na INTELovim chipovima registri duljine 80 bitova, rezultat je izračunat u proširenoj točnosti i onda kod spremanja u varijablu* `C` *zaokružen u jednostruki format.*

Do netočnosti u ostalim algoritmima dolazi u prvom redu zbog greške u varijabli `z`*. Iako je greška tek u zadnjoj decimali, potenciranjem na visoku potenciju, prisutna*

greška se može vrlo povećati. Npr. za $n = 100000$, $\epsilon = 2^{-24}$ *imamo* $(1+\epsilon)^n \approx 1 + n\epsilon = 1.005960464477539$. *S druge strane, za* $n = 1000000$ *imamo čak* $z = 1 + 0.05/10^6 = 1 + 5 \times 10^{-8}$, *pa je* $f\ell(z) = 1$ *što se vidi iz zadnjeg retka tabele. Dakle, do greške dolazi jer funkcija* `pot`: $z \mapsto z^n$ *nije dobro uvjetovana za veliko* n. *Provjerimo to formulom za uvjetovanost:*

$$\kappa(\texttt{pot}) = \frac{\mid z \cdot nz^{n-1} \mid}{\mid z^n \mid} = n\,.$$

Još jedan fenomen je zanimljiv. U prvom stupcu je rezultat za $n = 10000$ *točniji nego u drugom stupcu, dok je za* $n = 100000$ *manje točan. To dolazi od toga, što su greške zaokruživanja i pozitivne i negativne, pa u rezultatu može biti stvarna greška mnogo manja od očekivane. Gornja međa za grešku kod funkcije* `pow` *će biti manja nego kod uobičajenog potenciranja* $\texttt{pot}(z,n) = z \cdot z \cdots z$, *ali stvarna greška može biti manja kod* $\texttt{pot}(z,n)$ *nego kod* $\texttt{pow}(z,n)$. ■

Primjer 2.4.2. *Izračunajmo aproksimaciju broja e koristeći definiciju*

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n\,,$$

tako da se kao n *uzmu potencije proja* 10. *Sljedeća tabela je izračunata korištenjem jezika FORTRAN 77 u jednostrukoj preciznosti* ($\epsilon = 2^{-24}$):

n	e_n	$\mid e_n - e \mid$
10^1	2.593743	$1.24539D-01$
10^2	2.704811	$1.34705E-02$
10^3	2.717051	$1.23104E-03$
10^4	2.718597	$3.15107E-04$
10^5	2.721962	$3.68039E-03$
10^6	2.595227	$1.23055E-01$
10^7	3.293968	$5.75686E-01$

, $\qquad e_n = \left(1 + \frac{1}{n}\right)^n$.

Kako dolazi do greške? Uočimo da svaka negativna potencija od 10 *ima grešku jer u binarnom prikazu ima beskonačni prikaz. Kad se formira* $1 + 1/n$ *za veće* n *(npr. za* $n = 10^6$ *ili* 10^7*), samo nekoliko značajnih binarnih znamenaka od* $1/n$ *ostaje u konačnom prikazu od* $1 + 1/n$. *Dalje potenciranje tog broja* $1 + 1/n$, *makar izračunato posve točno, mora bitno povećati polaznu grešku. Lako se provjeri da računanje u dvostrukoj preciznosti* ($\epsilon = 2^{-53}$) *potencija polaznih aproksimacija od* $1 + 1/n$ *(koje su prvo smještene kao tipovi u jednostrukoj preciznosti, a onda konvertirani u dvostruku preciznost) daje iste brojeve u tabeli. Zaključujemo da za netočan rezultat nije kriv veliki broj računskih operacija već jedna podmukla greška.*

Ovaj problem, kao i prethodni, može se riješiti mnogo točnije korištenjem formule $(1 + 1/n)^n = exp(n \cdot ln(1 + 1/n))$, *pri čemu se* $ln(1 + 1/n)$ *računa tako da se*

prvo izračuna $y = 1 + x$, *a onda onda vrijednost funkcije*

$$f(x) = \begin{cases} x, & y = 1 \\ \frac{x ln(x)}{y-1}, & y \neq 1 \end{cases}.$$

Detaljna analiza grešaka zaokruživanja pokazuje da će $fl\,(f(x))$ *biti vrlo točna aproksimacija od* $ln(1 + 1/n)$. ■

Primjer 2.4.3. *Za* $x \geq 0$ *sljedeći algoritam ne bi smio promijeniti* x*:*

za i=1:60 računaj $x = \sqrt{x}$*;*
za i=1:60 računaj $x = x^2$*;*

Jer računanje ne uključuje oduzimanje i svi međurezultati leže između 1 *i* x*, možemo očekivati da je na kraju računanja, izračunati* $\hat{x}$ *dobra aproksimacija od* x*. Na kalkulatoru HP48G, polazeći od* $x = 100$*, algoritam daje* $\hat{x} = 1.0$*. Zapravo za svaki* x*, kalkulator izračuna umjesto* $f(x) = x$*, funkciju*

$$\hat{f}(x) = \begin{cases} 0\,, & 0 \leq x < 1 \\ 1\,, & x \geq 1 \end{cases}.$$

Dakle, kalkulator daje posve pogrešni rezultat na bazi samo 120 operacija, od kojih svaka ponaosob ima malu relativnu grešku. Zašto je tome tako?

Reprezentabilni brojevi na kalkulatoru HP48G zadovoljavaju nejednakost

$$10^{-499} \leq x \leq 9.99999999999 \cdot 10^{499}.$$

Definirajmo funkciju

$$r(x) = x^{\frac{1}{2^{60}}} \quad za\ x \geq 1\,,$$

koja odgovara međurezultatu nakon prve petlje. Vrijedi

$$\begin{aligned} 1 \leq r(x) &< r(10^{500}) = 10^{\frac{500}{2^{60}}} \approx e^{500 \cdot 2^{-60} ln(10)} \\ &< e^{10^{-15}} = 1 + 10^{-15} + \frac{1}{2}10^{-30} + \cdots \end{aligned}$$

pa se u kalkulatoru koji radi na 12 *decimala* $r(x)$ *zaokruži na* 1*. U drugoj petlji* 60 *kvadriranja jedinice opet daje jedinicu.*

Za $0 < x < 1$ *imamo* $x \leq 0.999999999999$ *jer je* x *reprezentabilan. Stoga za* $\sqrt{x}$ *vrijedi*

$$\begin{aligned} \sqrt{x} &\leq \sqrt{1 - 10^{-12}} = 1 - \frac{1}{2}10^{-12} - \frac{1}{8}10^{-24} - \cdots \\ &= 0.99999999999949999999999987499\ldots \end{aligned}$$

Ova gornja granica se zaokružuje na broj 0.999999999999. *I nakon* 60 *vađenja drugog korijena u kalkulatoru će biti broj koji ni veći od* 0.999999999999. *Promotrimo sada kvadriranja. Neka je* $s(x) = x^{2^{60}}$. *Imamo*

$$\begin{aligned} s(x) &\leq s(0.999999999999) = (1 - 10^{-12})^{2^{60}} = 10^{2^{60}\log(1-10^{-12})} \\ &= 10^{2^{60}\ln(1-10^{-12})\log(e)} \approx 10^{-2^{60}\cdot 10^{-12}\log(e)} \approx 3.568 \cdot 10^{-500708}\,. \end{aligned}$$

Jer je gornja granica manja od najmanjeg reprezentabilnog broja na kalkulatoru, isto vrijedi i za svaki $0 < x \leq 1 - 10^{-12}$. *Zato dolazi do zamjene broja s nulom (tzv.* underflow*).*

Zaključak je jasan: ništa nije loše u kalkulatoru. Ovo na prvi pogled nedužno računanje iscrpljuje preciznost i rang brojeva u mašini (12 značajnih dekadskih znamenki i eksponent s tri dekadske znamenke. ■

Primjer 2.4.4. *Poznato je da vrijedi*

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots = \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \approx 1.644934066848\,.$$

Pretostavimo da nismo svjesni tog indentiteta i da želimo numerički izračunati tu sumu. Njajednostavnija strategija je izračunati parcijalne sume $s_n = \sum_{k=1}^{n} \frac{1}{k^2}$ *sve dok se vrijednost od* s_n *ne ustali. U FORTRANU 77, u jednostrukoj preciznosti dobiva se* 1.64472532 *za* $n = 4096$. *Ta vrijednost se slaže sa* $\pi^2/6$ *tek u prvih* 4 *značajnih znamenki. Objašnjenje tako netočnog rezultata leži u činjenici da se zbrajanje vrši od većih prema manjim članovima reda. Za* $n = 4096$ *čini se sljedeći doprinos parcijalnoj sumi:*

$$s_n = s_{n-1} + \frac{1}{n^2} = s_{n-1} + 2^{-24}\,,$$

pri čemu je $s_{n-1} \approx 1.6$. *U jednostrukoj preciznosti računalo radi s mantisama od* 24 *bita pa član koji se dodaje ne daje nikakav doprinos, isto kao i svi sljedeći članovi.*

Najjednostavniji način za popravku te netočnosti je zbrajanje u obrnutom redoslijedu. Nažalost taj pristup zahtijeva znanje koliko članova zbrojiti. S 10^9 *članova dobije se suma* 1.66493406 *koja je korektna na* 8 *značajnih znamenaka.* ■

Detaljnija analiza o tome kako što točnije računati sumu brojeva može se naći u [2, 4. pogl.].

2.4.4. Kraćenje

Kraćenje nastaje kad se oduzimaju dva približno jednaka broja. To najčešće, iako ne uvijek, ima lošu posljedicu. Promotrimo npr. funkciju $f(x) = (1-\cos(x))/x^2$. Za $x = 1.2 \cdot 10^{-5}$ vrijednost od $\cos(x)$ zaokružena na 10 značajnih znamenki iznosi

$$c = 0.9999\,9999\,99,$$

tako da je

$$1 - c = 0.0000\,0000\,01.$$

Dakle je $(1-c)/x^2 = 10^{-10}/1.44^{-10} = 0.6944\ldots$, što je očito loše jer je $0 \leq f(x) \leq 1/2$ za sve $x \neq 0$. Dakle desetoroznamenkasta aproksimacija za $\cos(x)$ nije dovoljna da dade vrijednost od $f(x)$ s barem jednom točnom znamenkom. Problem je u tome što $1-c$ ima samo jednu značajnnu znamenku. Oduzimanje $1-c$ je *egzaktno*, ali to oduzimanje proizvodi rezulat koji je veličine kao i greška u c. Drugim riječima oduzimanje podiže značaj prethodne greške. U ovom primjeru $f(x)$ se može napisati tako da se izbjegne kraćenje. Jer je $\cos(x) = 1 - 2\sin^2(x/2)$, imamo

$$f(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

Izračunavanje $f(1.2 \cdot 10^{-5})$ pomoću ove druge formule korištenjem desetoroznamenkaste aproksimacije za $\sin(x/2)$ daje vrijednost 0.5, koja je točna na deset značajnih znamenaka.

Da bi dobili dodatni uvid u fenomen kraćenja razmotrimo oduzimanje (u egzaktoj aritmetici) $\hat{x} = \hat{a} - \hat{b}$, gdje su $\hat{a} = a(1+\Delta_a)$, $\hat{b} = b(1+\Delta_b)$. Članovi Δ_a i Δ_b su relativne greške ili netočnosti u podacima, koje recimo dolaze od prethodnih računanja. Izraz za relativnu grešku daje

$$\left|\frac{x-\hat{x}}{x}\right| = \left|\frac{-a\Delta_a + b\Delta_b}{a-b}\right| \leq \max\{|\,\Delta_a\,|, |\,\Delta_b\,|\}\frac{|\,a\,|+|\,b\,|}{|\,a-b\,|}.$$

Ograda za relativnu grešku od $\hat{x}$ je velika ako je $|\,a-b\,| \ll |\,a\,|+|\,b\,|$, a to je istina ako postoji jako kraćenje u oduzimanju. Ova analiza pokazuje da se zbog jakog kraćenja (u oduzimanju), posojeće greške ili netočnosti u $\hat{a}$ i $\hat{b}$ povečaju. Drugim riječima, *kraćenje dovodi prethodne greške na vidjelo.*

Važno je znati da kraćenje nije *uvijek* loša stvar. Ima nekoliko razloga. Prvo, brojevi koji se oduzimaju mogu biti bez prethodnih grešaka, dakle točni ulazni podaci. Npr. računanje podijeljenih razlika uključuje mnogo oduzimanja, ali pola tih oduzimanja uključuje polazne podatke, pa su bezopasna za pogodni uređaj točaka. Drugi je razlog što kraćenje može biti simptom loše uvjetovanosti problema pa zato mora biti prisutno. Treće, efekt kraćenja ovisi o ulozi koji taj rezultat (zapravo međurezultat) igra u preostalom računanju. Npr. ako je $x \gg y \approx z > 0$, tada je kraćenje u izrazu $x + (y - z)$ bezopasno.

Promotrimo jedan važan primjer kraćenja koji je karakterističan za područje numeričkog rješavanja diferencijalnih jednadžbi.

Primjer 2.4.5. *Kako odrediti približnu vrijednost derivacije neke realne funkcije realnog argumenta. Ako funkciju označimo sa f, tada je*

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

U xy koordinatnum sustavu, derivacija ima smisao koeficijenta smjera pravca koji je tangenta u točki $(x, f(x))$ na graf funkcije f tj. tangens kuta kojeg pravac zatvara s apscisom). U ovoj definiciji h može biti i pozitivan i negativan (osim ako se promatra tzv. derivacija s lijeva ili s desna za rubne vrijednosti od x). Što je h po modulu manji to će kvocijent $(f(x+h)-f(x))/h$ biti točnija aproksimacija od $f'(x)$. Za funkcije koje nisu zadane analitički (svojom formulom), nego nekim možda kompliciranim algoritmom ili tabličnim vrijednostima, kvocijent $(f(x+h)-f(x))/h$ koji se još zove podijeljena razlika *ili* podijeljena diferencija *moći će se izračunati za barem neke male vrijednosti od h, pa ćemo imati neku informaciju o derivaciji. Što ćemo korak ili pomak h moći manjim odabrati, to ćemo točniju aproksimaciju od $f'(x)$ dobiti.*

Pretpostavimo sada da je f dva puta diferencijabilna. Koristeći Taylorov razvoj funkcije f oko točke x, dobijemo

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\bar{x}),$$

gdje je $\bar{x} \in [x, x+h]$ ako je $h \geq 0$ i $\bar{x} \in [x+h, x]$ ako je $x < 0$. To nam daje ocjenu

$$\frac{f(x+h)-f(x)}{h} - f'(x) = \frac{h^2}{2} f''(\bar{x}) \tag{2.4.2}$$

pa možemo očekivati da je aproksimacija to bolja što je $\mid h \mid$ manje. Greška (2.4.2) se zove diskretizacijska greška. ■

Što se međutim događa ako koristimo konačnu aritmetiku?

Primjer 2.4.6. *Odaberimo funkciju koja je eksplicitno zadana i kod koje znamo i izraz za derivaciju. Neka je to npr.* $\mathrm{ch}(x) = (e^x + e^{-x})/2$. *Derivacija je funkcija* $\mathrm{sh}(x) = (e^x - e^{-x})/2$. *Promotrimo aproksimacije te funkcije u $x = 1$ na računalu, koristeći dvostruki IEEE format. Kako je* $\mathrm{ch}(1) = (e^2+1)/2e \approx 1.5431$ *i* $\mathrm{ch}'(1) = \mathrm{sh}(1) = (e^2-1)/2e \approx 1.1752$, *znamo vrijednosti funkcije i derivacije u točki* 1. *Za h ćemo uzeti padajući niz vrijednosti* $0.1, 0.01, 0.001, \ldots, 10^{-20}$. *Za računaje izraza $(f(x+h)-f(x)/h$ koristit ćemo sljedeći niz naredbi*

```
x:=1.0;  fx:=f(x); der:=(exp(x)-exp(-x))/2.0;
for i:=1 to 17
begin h:=10^(-i); xh:=x+h; fh:=f(xh);
      dif:=f(xh)-f(x); g:=dif/h;
      apsgr:= g-der; relgr:=(g-der)/der
end
```

gdje je `f(x)=(exp(x)+exp(-x))/2`. *Vrijednosti koje smo dobili korištenjem analognih naredbi u jeziku FORTRAN 77, uz Digital Visual Fortran compiler 6.0 su dani u sljedećoj tabeli:*

h	der	g	apsgr	relgr
0.10E00	0.11752012E+01	0.12543792E+01	0.79E-01	0.67E-01
0.10E-01	0.11752012E+01	0.11829362E+01	0.77E-02	0.66E-02
0.10E-02	0.11752012E+01	0.11759729E+01	0.77E-03	0.66E-03
0.10E-03	0.11752012E+01	0.11752783E+01	0.77E-04	0.66E-04
0.10E-04	0.11752012E+01	0.11752089E+01	0.77E-05	0.66E-05
0.10E-05	0.11752012E+01	0.11752020E+01	0.77E-06	0.66E-06
0.10E-06	0.11752012E+01	0.11752013E+01	0.78E-07	0.67E-07
0.10E-07	0.11752012E+01	0.11752012E+01	0.94E-08	0.80E-08
0.10E-08	0.11752012E+01	0.11752013E+01	0.76E-07	0.65E-07
0.10E-09	0.11752012E+01	0.11752022E+01	0.96E-06	0.82E-06
0.10E-10	0.11752012E+01	0.11752155E+01	0.14E-04	0.12E-04
0.10E-11	0.11752012E+01	0.11752821E+01	0.81E-04	0.69E-04
0.10E-12	0.11752012E+01	0.11746160E+01	-0.59E-03	-0.50E-03
0.10E-13	0.11752012E+01	0.11768364E+01	0.16E-02	0.14E-02
0.10E-14	0.11752012E+01	0.13322676E+01	0.16E+00	0.13E+00
0.10E-15	0.11752012E+01	0.00000000E+00	-0.12E+01	-0.10E+01
0.10E-16	0.11752012E+01	0.00000000E+00	-0.12E+01	-0.10E+01

Iz tabele vidimo da su aproksimacije sve bolje do određene vrijednosti od h, koje je oko $\sqrt{\epsilon_M}$. Apsolutna i relativna greška smanjuju se za faktor 10 što je sukladno formuli (2.4.2). Međutim, kad h postaje sve manji, vrijednosti $f(x+h)$ i $f(x-h)$ postaju sve bliskije, pa dolazi do jakog kraćenja. Npr. za $h = 10^{-10}$ imamo $f(x+h) - f(x) \approx 1.543080634932764 - 1.543080634815244 = 1.1752 \cdot 10^{-10}$, pa je izgubljeno 10 decimalnih znamenaka, što znači da relativna greška aproksimacije ne može biti bitno bolja od 10^{-6}. Svakim smanjivanjem koraka za 10, pojačava se kraćenje za jednu decimalnu znamenku, što rezultira relativnom greškom koja je za približno faktor 10 veća. Apsolutna greška je istog reda veličine jer se dobije iz relativne greške množenjem sa $\mid f'(x) \mid \approx 1.1752$. ■

Dokle god je f dovoljno glatka, postoji mogućnost dobivanja bolje aproksimacije od $f'(x)$ nego pomoću obične podijeljene razlike. Jedna takova je *centralna podijeljena razlika*

$$\frac{f(x+h) - f(x-h)}{2h}.$$

Zadatak 2.4.1. *Razvojem u Taylorov red $f(x+h)$ i $f(x-h)$ oko točke x, pokažite da vrijedi*

$$\frac{f(x+h) - f(x-h)}{2h} - f'(x) = \frac{h^2}{12}\left(f'''(\bar{x}_1) + f'''(\bar{x}_2)\right) \tag{2.4.3}$$

gdje su $\bar{x}_1$ i $\bar{x}_2$ u intervalima $[x, x+h]$ i $[x-h, x]$. Dakle, diskretizacijska greška pada s faktorom h^2. Načinite program sličan prethodnom, koji ispisuje apsolutnu i relativnu grešku za centralnu podijeljenu razliku kao aproksimaciju derivacije za $f(x) = \mathrm{ch}(x)$ u točki $x = 1$. Rezultat bi trebala biti tabela oblika

h	der	g	apsgr	relgr
0.10E+00	0.11752012E+01	0.11771608E+01	0.20E-02	0.17E-02
0.10E-01	0.11752012E+01	0.11752208E+01	0.20E-04	0.17E-04
0.10E-02	0.11752012E+01	0.11752014E+01	0.20E-06	0.17E-06
0.10E-03	0.11752012E+01	0.11752012E+01	0.20E-08	0.17E-08
0.10E-04	0.11752012E+01	0.11752012E+01	0.16E-10	0.13E-10
0.10E-05	0.11752012E+01	0.11752012E+01	-0.62E-10	-0.53E-10
0.10E-06	0.11752012E+01	0.11752012E+01	-0.62E-09	-0.53E-09
0.10E-07	0.11752012E+01	0.11752012E+01	-0.17E-08	-0.15E-08
0.10E-08	0.11752012E+01	0.11752012E+01	-0.35E-07	-0.30E-07
0.10E-09	0.11752012E+01	0.11752010E+01	-0.15E-06	-0.12E-06
0.10E-10	0.11752012E+01	0.11752044E+01	0.32E-05	0.27E-05
0.10E-11	0.11752012E+01	0.11751711E+01	-0.30E-04	-0.26E-04
0.10E-12	0.11752012E+01	0.11746160E+01	-0.59E-03	-0.50E-03
0.10E-13	0.11752012E+01	0.11768364E+01	0.16E-02	0.14E-02
0.10E-14	0.11752012E+01	0.12212453E+01	0.46E-01	0.39E-01
0.10E-15	0.11752012E+01	0.00000000E+00	-0.12E+01	-0.10E+01
0.10E-16	0.11752012E+01	0.00000000E+00	-0.12E+01	-0.10E+01

Uvjerite se da greške padaju s faktorom 100, a ne 10 kako je to bilo prije. Nakon vrijednosti `h=.10E-05` greke rastu kao i prije s faktorom oko 10. Za to vrijdi isti argument kao u prethodnm primjeru. Koji argument biste dali za činjenicu da je minimalna greška manja nego u prethodnom slučaju? ■

2.4.5. Kraćenje grešaka zaokruživanja

Nije neobično da se u stabilnim algoritmima greške zaokruživanja krate na taj način da konačni rezultat bude mnogo točniji od međurezultata (tj. veličina koje se koriste u algoritmu). Ovaj fenomen nije toliko poznat jer se obično međurezultati gledaju tek kad se otkrije da krajnji rezultat nije dovoljno točan. Evo jednog primjera

Promotrimo računanje funkcije

$$f(x) = \frac{e^x - 1}{x} = 1 + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \cdots = \sum_{i=0}^{\infty} \frac{x}{(i+1)!},$$

koja se pojavljuje u raznim aplikacijama. Pogledajmo sljedeća dva algoritma koji računaju tu funkciju.

```
c   Algoritam 1

    y=exp(x)
    if (x.eq.0.0)then
      f=1.0
    else
      f=(y-1.0)/x
    endif
```

```
c   Algoritam 2

    y=exp(x)
    z=alog(y)
    if (y.eq.1.0)then
      f=1.0
    else
      f=(y-1.0)/z
    endif
```

Prvi, ima lošu stranu što dolazi do jakog kraćenja za male vrijednosti od $|x|$. Drugi algoritam, izgleda čudno jer izračunava dvije funkcije exp i ln umjesto samo jedne exp, kao što čini prvi algoritam.

Međutim ako se pogledaju rezulati za $x = 10^{-k}$, $k = 0, 1, \ldots, 15$ u Tablici 2.4.5., vidi se da Algoritam 2 daje za sve x vrlo točne rezultate dok Algoritam 1 daje za rastuće k sve netočnije podatke.

x	Algoritam 1	Algoritam 2	Egzaktno
0.1000000E+00	0.1051710E+01	0.1051709E+01	0.1051709E+01
0.1000000E-01	0.1005018E+01	0.1005017E+01	0.1005017E+01
0.1000000E-02	0.1000524E+01	0.1000500E+01	0.1000500E+01
0.1000000E-03	0.1000166E+01	0.1000050E+01	0.1000050E+01
0.1000000E-04	0.1001358E+01	0.1000005E+01	0.1000005E+01
0.1000000E-05	0.9536742E+00	0.1000000E+01	0.1000000E+01
0.1000000E-06	0.1192093E+01	0.1000000E+01	0.1000000E+01
0.1000000E-07	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-08	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-09	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-10	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-11	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-12	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-13	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-14	0.0000000E+00	0.1000000E+01	0.1000000E+01
0.1000000E-15	0.0000000E+00	0.1000000E+01	0.1000000E+01

Tablica 2.4.1. Primjena algoritama 1 i 2 za $x = 10^{-k}$, $k = 0, 1, \ldots, 15$

Da bi dobili uvid što se to događa s Algoritmom 2 promotrimo računanje za $x = 9{\cdot}10^{-8}$ uz $\epsilon = 2^{-24} \approx 6{\cdot}10^{-8}$. Točni je rezultat (do na broj prikazanih znamenki) 1.000000005. Algoritam 1 daje posve netočan rezultat, kao što se i očekuje

$$f\ell\left(\frac{e^x - 1}{x}\right) \equiv f\ell\left(\frac{1.19209290 \cdot 10^{-7}}{9.0000000 \cdot 10^{-8}}\right) = 1.32454766\,.$$

Algoritam 2 daje rezultat koji je točan u svim osim u zadnjoj znamenci (što se i očekuje jer se radi o 9. značajnoj znamenki, a aritmetika radi s nepunih 8 znamenaka)

$$f\ell\left(\frac{e^x-1}{\ln(x)}\right) \equiv f\ell\left(\frac{1.19209290\cdot 10^{-7}}{1.19209282\cdot 10^{-7}}\right) = 1.00000006\,.$$

Evo sada veličina koje bi dobili Algoritmom 2 u egzaktoj aritmetici (korektnih na onoliko decimala koliko se koristi u prikazu)

$$\frac{e^x-1}{\ln(x)} = \frac{9.00000041\cdot 10^{-8}}{9.00000001\cdot 10^{-8}} = 1.00000005\,.$$

Vidimo da Algoritam 2 izračunava vrlo netočno vrijednosti e^x-1 i $\ln(x)$, ali je kvocijent tih vrijednosti vrlo točan rezultat. Zaključak: u dijeljenju kod Algoritma 2 greške se pokrate (dokinu). Koristeći analizu grešaka zaokruživanja može se objasniti tu začuđujuću pokratu grešaka (vidjeti [2, str.]).

2.4.6. Rješavanje kvadratne jednadžbe

Rješavanje kvadratne jednadžbe je $ax^2+bx+c=0$ je matematički trivijalno: ako je $a\neq 0$, postoje dva rješenja

$$x_{\pm} = \frac{-b\pm\sqrt{b^2-4ac}}{2a}. \tag{2.4.4}$$

Ako je $a=0$, postoji jedno rješenje $x=-c/b$ linearne jednadžbe.

Numerički, problem je izazovniji, jer niti izračunavanje izraza (2.4.4), niti točnost izračunatog rješenja nije garantirana.

Najjednostavniji aspekt rješavanja kvadratne jednadžbe je izbor formule za računanje rješenja. Ako je $b^2 \gg |\,4ac\,|$ tada je $\sqrt{b^2-4ac}\approx|\,b\,|$, pa se za jedan izbor predznaka u formuli (2.4.4) događa kraćenje. To je opasno kraćenje jer je jedan od argumenata, $f\ell(\sqrt{b^2-4ac})$ netočan zbog korištenja konačne aritmetike, pa kraćenje pojačava grešku u $f\ell(\sqrt{b^2-4ac})$ (također i u $f\ell(b)$ ako i on nosi grešku u odnosu na b). Kako izbjeći grešku je dobro poznato: treba prvo izračunati veće (po modulu) rješenje, nazovimo ga s x_1, pomoću formule

$$x_1 = \frac{-(b+\mathrm{sign}(b)\sqrt{b^2-4ac})}{2a}, \tag{2.4.5}$$

dok se drugo rješenje dobije iz formule $x_1x_2=c/a$,

$$x_2 = \frac{c}{a\cdot x_1} = \frac{-2c}{b+\mathrm{sign}(b)\sqrt{b^2-4ac}}, \tag{2.4.6}$$

pri čemu se obje formule u (2.4.6) mogu koristiti. Formula $c/(a \cdot x_1)$ je vrlo točna, tj. ako se x_1 izračuna s relativnom greškom ε_{x_1}, tada se x_2 izračuna s relativnom greškom ne većom od $\varepsilon_{x_1} + 2\epsilon$. Druga formula za x_2 u (2.4.6) je direktnija jer koristi samo koeficijente kvadratne jednadžbe, ali zato ima nešto više operacija od prve. Ocjena greške za x_2 po toj formuli će biti jednaka ocjeni greške za x_1 po formuli (2.4.5). Stoga se možemo posvetiti proučavanju točnosti formule (2.4.5) za x_1.

Nažalost postoji mnogo opasniji izvor kraćenja, onaj u izrazu $b^2 - 4ac$. Točnost se gubi ako je $b^2 \approx 4ac$ (slučaj bliskih korijena) i nikakva algebarska transformacija neće izbjeći kraćenje. Jedini način da se garantira točno izračunata diskriminanta je korištenje povećane preciznosti (ili trikova u postojećoj aritmetici koji su ekvivalentni u rezultatu, korištenju dvostruke preciznosti) u izračunavanju izraza $b^2 - 4ac$.

Zbog važnosti koju kvadratna jednadžba ima u numeričkom računanju, cjelovitu analizu grešaka koje nastaju kod traženja korijena ćemo načiniti kad se upoznamo sa osnovama analize grešaka zaokruživanja.

Umjesto zaključka pokušat ćemo ukratko dati upute kako dizajnirati stabilne (tj. točne u okruženju strojne aritmetike) algoritme te pokušati svratiti pažnju na često prisutna kriva uvjerenja vezana uz računanje na računalima. Neke od tvrdnji koje ćemo sada susresti obrazložit ćemo u sljedećim poglavljima jer se tiču specijalnih područja numeričke matematike.

2.4.7. Kako dizajnirati stabilne algoritme

Nema jednostavnog recepta za dizajniranje stabilnih algoritama, ali najbolji je savjet spoznaja da je numerička stabilnost važnija od drugih karakteristika algoritma, kao što su npr. broj računskih operacija, te dobra vektorizacija ili paralelizacija (prilagodljivost algoritma vektorskim ili višeprocesorskim računalima). Evo nekih općih uputa u redosljedu kako se spominju u [2]:

1. Pokušajte izbjeći oduzmanje veličina bliskih vrijednosti koje nose greške, iako je to kadkad nemoguće.
2. Minimizirajte veličinu međurezultata u odnosu na konačni rezultat. To je važno da se konačni rezultat ne bi dobio kao opasno oduzimanje velikih vrijednosti koje nose u sebi greške.
3. Potražite drugačije formulacije istog problema ili druge formule za isti račun.
4. Koristite prednosti jednostavnih formula za ažuriranje tipa

   $$\text{nova vrijednost} = \text{stara vrijednost} + \text{mala korekcija}$$

   ako se mala korekcija može izračunati na dovoljan broj značajnih znamenaka (Eulerova metoda, Newtonova metoda,...).

5. Koristite samo dobro uvjetovane transformacije za dobivanje rješenja. Kod matrica to znači koristiti ortogonalne matrice kad god je to moguće, jer neortogonalne matrice mogu biti loše uvjetovane.
6. Preduzeti mjere opreza da se spriječe moguća prekoračenja granica brojeva s kojima radi konačna aritmetika (*overflow* i *underflow*).
7. Kod nekih računala centralna aritmetička jedinica koristi precizniju aritmetiku za operande u registrima, a zaokruživanje nastupa tek kod spremanja podatka u memoriju. To znači da nije povoljno cijepati složenije formule u više programskih linija koristeći pomoćne varijable.

U tijeku računanja uvijek je dobro imati određeni monitoring nekih po stabilnost važnih veličina. Ako se u tijeku računanja pamti i ažurira najveći po modulu broj on može biti indikacija što se događa s algoritmom.

S vremenom se nakupilo podosta krivih uvjerenja u vezi računanja na računalima. Nabrojimo neka

1. Kraćenje pri oduzimanju je uvijek loša stvar
2. Greške zaokruživanja mogu "nadvladati" rezultat samo ako ih ima veliki broj.
3. Kratki račun bez kraćenja i bez prekoračenja granica (*underflow* i *overflow*) uvijek mora dati točan rezultat (vidjeti [2]).
4. Povećanje preciznosti aritmetike koja se koristi uvijek povećava točnost rezultata (vidjeti [2]).
5. Izlazni rezultat kod nekog algoritma ne može biti točniji od bilo kojeg međurezultata, tj. greške se ne mogu pokratiti.
6. Greške zaokruživanja mogu samo pogoršati, ali ne i pomoći u uspjehu računanja. Ovdje je tipični primjer metoda inverznih iteracija za računanje vlastitih vektora matrice (vidjeti [4, 2]).

2.5. Osnove analize grešaka zaokruživanja

U ovom dijelu, pozabavit ćemo se osnovnim alatom za analiziranje stabilnosti numeričkih algoritama. Analiza grešaka zaokruživanja je zajedno sa perturbacijskom analizom broblema koji se rješava, moćan alat za analiziranje pa zato i dizajniranje numeričkih algoritama.

Označimpo sa $\mathtt{P}$ skup negativnih potencija broja 2 koje ulaze u definiciju strojne preciznosti u IEEE standardu:

$$\mathtt{P} = \{2^{-23}, 2^{-24}, 2^{-52}, 2^{-53}, 2^{-63}\}. \tag{2.5.1}$$

Ako je n cijeli broj koji ima ulogu dimenzije vektora ili matrice, stupnja polinoma, broja sumanada u konačnoj sumi ili broja faktora u konačnom produktu i sl., tada ćemo pretpostaviti da vrijedi

$$\epsilon \in \mathtt{P} \Rightarrow n\epsilon \leq 2^{-6}\,. \tag{2.5.2}$$

To znači, ako radimo u jednostrukoj preciznosti i koristimo bilo koji način zaokruživanja, maksimalna vrijednost od n će biti $2^{17} = 131072$. Ako koristimo standardni način zaokruživanja do najbližeg, dozvoljavamo $n \leq 262144$. Ako želimo raditi sa još većim dimenzijama, pretpostavka je da ćemo koristiti dvostruku preciznost (IEEE dvostruki format). U tom slučaju, za n dozvoljavamo gornju granicu $2^{46} \approx 7.037 \cdot 10^{13}$, a ako još koristimo standardni način zaokruživanja, n može bit velik kao $1.40737488355328 \cdot 10^{14}$. Ako želimo još veći n ili $-n$, moramo osigurati korištenje računanja u proširenoj točnosti. Konačno, ako u razmatranje želimo uključiti i kalkulatore, tada u skup $\mathtt{P}$ možemo ubaciti i neke negativne potencije od 10 (npr. 10^{-12}), jer kalkulatori koriste decimalnu aritmetiku.

Sljedeća "tehnička" lema će se često koristiti u ocjenjivanju grešaka zaokruživanja.

Lema 2.5.1. *Neka $\epsilon \in \mathtt{P}$ i n zadovoljavaju uvjet (2.5.2). Ako je $\mid \varepsilon \mid \leq \epsilon$, tada vrijedi*

(i)	$(1+\varepsilon)^2 = 1+\varepsilon_2,$	$\mid \varepsilon_2 \mid \leq 2.00000012\epsilon$
(ii)	$(1+\varepsilon)^3 = 1+\varepsilon_3,$	$\mid \varepsilon_3 \mid \leq 3.00000036\epsilon$
(iii)	$(1+\varepsilon)^{-1} = 1+\varepsilon_1',$	$\mid \varepsilon_1' \mid \leq 1.00000012\epsilon$
(iv)	$(1+\varepsilon)^{-2} = 1+\varepsilon_2',$	$\mid \varepsilon_2' \mid \leq 2.00000036\epsilon$
(v)	$(1+\varepsilon)^n = 1+\varepsilon_n,$	$\mid \varepsilon_n \mid \leq 1.008n\epsilon$
(vi)	$(1+\varepsilon)^{-n} = 1+\varepsilon_n',$	$\mid \varepsilon_n' \mid \leq 1.008n\epsilon$
(vii)	$(1+\varepsilon)^{\frac{1}{2}} = 1+\varepsilon_{\surd},$	$\mid \varepsilon_{\surd} \mid \leq \frac{1}{2} \mid \varepsilon \mid \leq \frac{1}{2}\epsilon.$

Dokaz. (i) Zbog $(1+\varepsilon)^2 = 1+\varepsilon(2+\varepsilon)$ je

$$\mid \varepsilon_2 \mid = \mid 2+\varepsilon \mid \cdot \mid \varepsilon \mid \leq (2+2^{-23})\epsilon \leq 2.00000012\epsilon.$$

(ii) Slično, $(1+\varepsilon)^3 = 1+\varepsilon(3+3\varepsilon+\varepsilon^2)$, pa je

$$\mid \varepsilon_3 \mid \leq (3+3\cdot 2^{-23}+2^{-46})\epsilon \leq 3.00000036\epsilon.$$

(iii) Jer je $\mid \varepsilon \mid < 1$, imamo

$$\left|\frac{1}{1+\varepsilon}\right| \leq \frac{1}{1-\mid \varepsilon \mid} = 1 + \frac{\mid \varepsilon \mid}{1-\mid \varepsilon \mid} \leq 1 + (1-2^{-23})^{-1}\epsilon \leq 1.00000012\epsilon\,.$$

(iv) Korištenjem Taylorovog razvoja $(1+t)^{-2} = 1 + 2t + 3t^2 + \cdots kt^{k-1} + \cdots$, odmah slijedi $(1+\varepsilon)^{-2} = 1 + \varepsilon(2 + 3\varepsilon + 4\varepsilon^2 + \cdots)$ i još treba iskoristiti da je $2 + 3|\varepsilon| + 4|\varepsilon|^2 + 5|\varepsilon|^3 + \cdots \leq 2.00000035762793$.

(v) Razvoj binoma na n-tu potenciju daje

$$\begin{aligned} |\,\varepsilon_n\,| &= n\,|\,\varepsilon(1 + \frac{n-1}{2}\varepsilon + \frac{(n-1)(n-2)}{2\cdot 3}\varepsilon^2 + \cdots + \varepsilon^{n-2} + \frac{1}{n}\varepsilon^{n-1})\,| \\ &\leq n\epsilon(1 + \frac{2^{-6}}{2}(1 + \frac{n-2}{3}\epsilon + [\frac{n-2}{3}\epsilon]^2 + [\frac{n-2}{3}\epsilon]^3 + \cdots) \\ &\leq n\epsilon(1 + \frac{2^{-7}}{1 - \frac{(n-2)\epsilon}{3}}) \leq n\epsilon(1 + \frac{2^{-7}}{1 - \frac{2^{-6}}{3}}) = \frac{385}{382}n\epsilon \leq 1.008n\epsilon\,. \end{aligned}$$

(vi) Dokaz je posve analogan prethodnom. Kako je zbog (iii) $(1+\varepsilon)^{-n} = (1+\varepsilon')^n = 1 + \varepsilon'_n$, imamo

$$\begin{aligned} |\,\varepsilon'_n\,| &= n\,|\,\varepsilon'(1 + \frac{n-1}{2}\varepsilon' + \frac{(n-1)(n-2)}{2\cdot 3}\varepsilon'^2 + \cdots + \varepsilon'^{n-2} + \frac{1}{n}\varepsilon'^{n-1})\,| \\ &\leq n\alpha\epsilon(1 + \frac{2^{-6}\alpha}{2}(1 + \frac{n-2}{3}\alpha\epsilon + [\frac{n-2}{3}\alpha\epsilon]^2 + \cdots) \\ &\leq n\alpha\epsilon(1 + \frac{2^{-7}\alpha}{1 - \frac{(n-2)\alpha\epsilon}{3}})\,, \end{aligned}$$

gdje je $\alpha = 1.00000012$. Kako je $\alpha(1 + \frac{2^{-7}\alpha}{1 - \frac{2^{-6}\alpha}{3}}) < 1.007854$.

(vii) Za svako $x > 0$ vrijedi

$$1 + \frac{x}{1+\sqrt{1+x}} = \sqrt{1+x} \leq 1 + \frac{1}{2}x \quad \text{i} \quad 1 - \frac{x}{1+\sqrt{1-x}} = \sqrt{1-x} \leq 1 - \frac{1}{2}x$$

pa za svako η i $\eta_{\surd}$ koje su povezane relacijom $\sqrt{1+\eta} = 1 + \eta_{\surd}$, vrijedi

$$\frac{|\,\eta\,|}{1+\sqrt{1+\bar{\eta}}} \leq |\,\eta_{\surd}\,| \leq \frac{1}{2}\,|\,\eta\,|\,, \quad \eta \leq \bar{\eta} \quad \text{i} \quad \operatorname{sign}(\eta_{\surd}) = \operatorname{sign}(\eta), \tag{2.5.3}$$

gdje je $\bar{\eta}$ proizvoljna gornja međa za η. Obje nejednakosti u (2.5.3) daju jaču tvrdnju od (vii) jer uz $\eta = \varepsilon$, $\bar{\eta} = \epsilon$ imamo i više od (vii). Time je dokaz lemme 2.5.1. je završen. ■

Za prvi dojam o korisnosti leme 2.5.1., riješimo sljedeći problem.

Primjer 2.5.1. *Neka su x i y reprezentabilni u računalu, tako da vrijedi $x = fl(x)$ i $y = fl(y)$. S kolikom relativnom greškom će računalo koje koristi IEEE standard izračunati z, gdje je $z = fl\sqrt{x^2 + y^2}$?*

Da bi riješili problem, pretpostavimo prvo da je

$$f\ell(x^2) + f\ell(y^2) < N_{max} \quad i \quad \min\{|\, x \,|, |\, y \,|\} \geq \sqrt{N_{min}}.$$

Tada, možemo pisati

$$\begin{aligned} x_2 &= x \otimes x = f\ell(x^2) = x^2(1+\varepsilon_1), \quad |\varepsilon_1| \leq \epsilon \\ y_2 &= y \otimes y = f\ell(y^2) = y^2(1+\varepsilon_2), \quad |\varepsilon_2| \leq \epsilon\,. \end{aligned}$$

Umjesto $f\ell(f\ell(x^2)+f\ell(y^2)) = f\ell(x^2) \oplus f\ell(y^2))$ *kraće se piše* $f\ell(x^2+y^2)$. *Sada imamo*

$$\begin{aligned} z_2 &= f\ell(x_2+y_2) = (x_2+y_2)(1+\varepsilon_3), \quad |\varepsilon_3| \leq \epsilon \\ z &= f\ell(\sqrt{z_2}) = \sqrt{z_2}(1+\varepsilon_4) \quad |\varepsilon_4| \leq \epsilon. \end{aligned}$$

Povežimo sve te jednadžbe.

$$\begin{aligned} z &= (1+\varepsilon_4)\sqrt{z_2} \\ &= (1+\varepsilon_4)\sqrt{(x_2+y_2)(1+\varepsilon_3)} \\ &= (1+\varepsilon_4)\sqrt{(1+\varepsilon_3)}\sqrt{x^2(1+\varepsilon_1)+y^2(1+\varepsilon_2)} \\ &= (1+\varepsilon_4)\sqrt{(1+\varepsilon_3)}\sqrt{(x^2+y^2}\sqrt{1+\frac{x^2\varepsilon_1+y^2\varepsilon_2}{x^2+y^2}} \\ &= (1+\varepsilon_4)\sqrt{1+\varepsilon_3}\sqrt{1+\varepsilon_5}\sqrt{(x^2+y^2}, \quad \varepsilon_5 = \frac{x^2\varepsilon_1+y^2\varepsilon_2}{x^2+y^2} \\ &= \sqrt{x^2+y^2}(1+\varepsilon_z), \quad 1+\varepsilon_z = (1+\varepsilon_4)\sqrt{(1+\varepsilon_3)(1+\varepsilon_5)}\,. \end{aligned}$$

Ocijenimo prvo ε_5. *Kako je*

$$\frac{x^2\varepsilon_1+y^2\varepsilon_2}{x^2+y^2} = \frac{x^2}{x^2+y^2}\varepsilon_1 + \frac{y^2}{x^2+y^2}\varepsilon_2,$$

ε_5 *je konveksna suma*[1] *brojeva* ε_1 *i* ε_2, *ona se nalazi u zatvorenom intervalu* $[\varepsilon_1, \varepsilon_2]$ *(ili* $[\varepsilon_2, \varepsilon_1]$ *ako je* $\varepsilon_2 < \varepsilon_1$*). To opet znači da vrijedi*

$$|\, \varepsilon_5 \,| \leq \max\{|\, \varepsilon_1 \,|\,,\ |\, \varepsilon_2 \,|\} \leq \epsilon\,.$$

Sada imamo,

$$\begin{aligned} |\, \varepsilon_z \,| &= |\, (1+\varepsilon_4)\sqrt{(1+\varepsilon_3)(1+\varepsilon_5)} - 1 \,| \\ &\leq (1+\epsilon)\sqrt{(1+\epsilon)(1+\epsilon)} - 1 \leq (1+\epsilon)(1+\epsilon) - 1 \\ &\leq (2+\epsilon)\epsilon \leq 2.00000012\epsilon\,. \end{aligned}$$

[1] suma $\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_k x_k$ je konveksna ako vrijedi $\sum_i \alpha_i = 1$, te za svako i $0 \leq \alpha_i \leq 1$.

Kod korištenja IEEE aritmetike, slučaj prekoračenja će biti dojavljen. Stoga programer treba ugraditi u kod programa grananje koje kod prekoračenja nastavlja s računanjem po formuli (vidi primjer 2.3.3.)

$$u = \max\{\mid x \mid\ ,\ \mid y \mid\}, \quad v = \min\{\mid x \mid\ ,\ \mid y \mid\}, \quad z = f\ell(u\sqrt{1+(v/u)^2})\,.$$

Ova formula je sigurna jer nam podatak $x = f\ell(x)$ i $y = f\ell(y)$ ukazuje da su u i v reprezentabilni brojevi. U ovom slučaju je prekoračenje moguće tek ako je $u \leq N_{max} < u\sqrt{1+(v/u)^2}$ $(\leq 2u)$. Pretpostavimo da to nije slučaj kao i da neće doći do potkoračenja međurezultata (o tome ćemo kasnije). Načinimo uz te pretpostavke analizu grešaka zaokruživanja za taj drugi algoritam.

Da bi analiza bila jasnija uvest ćemo niz pomoćnih varijabli z_i čije vrijednosti su reprezentabilni brojevi.

$$\begin{aligned}
z_1 &= f\ell(v/u) = (v/u)(1+\varepsilon_1), \quad \mid \varepsilon_1 \mid \leq \epsilon \\
z_2 &= f\ell(z_1^2) = (z_1^2)(1+\varepsilon_2), \quad \mid \varepsilon_2 \mid \leq \epsilon \\
z_3 &= f\ell(1+z_2) = (1+z_2)(1+\varepsilon_3), \quad \mid \varepsilon_3 \mid \leq \epsilon \\
z_4 &= f\ell(\sqrt{z_3}) = \sqrt{z_3}(1+\varepsilon_4), \quad \mid \varepsilon_4 \mid \leq \epsilon \\
z_5 &= f\ell(u \cdot z_4) = (u \cdot z_4)(1+\varepsilon_5), \quad \mid \varepsilon_5 \mid \leq \epsilon\,.
\end{aligned}$$

Postupak možemo nastaviti kao i prije počevši "od kraja":

$$z_5 = (u \cdot z_4)(1+\varepsilon_5) = u \cdot \sqrt{(1+z_2)(1+\varepsilon_3)}(1+\varepsilon_4)(1+\varepsilon_5) = \cdots$$

Pokušajte završiti taj postupak.

Da bi pokazali druge mogućnosti, ocijenimo krenuvši "od početka", relativne greške koje sadrže svi međurezultati. Zamislimo da se koristila egzaktna (još kažemo beskonačna) aritmetika. Tada bismo umjesto vrijednosti z_i dobili neke druge vrijednosti, nazovimo ih sa w_i, $1 \leq i \leq 5$. Ako pišemo

$$z_i = w_i(1+\eta_i), \quad 1 \leq i \leq 5,$$

tada trebamo ocijeniti η_5. Do među za $\mid \eta_5 \mid$ možemo doći ocjenjujući greške η_1, η_2, η_3 i η_4 u redosljedu kako su napisane. S obzirom da je $\eta_1 = \varepsilon_1$, imamo $\mid \eta_1 \mid \leq \epsilon$. Nadalje, imamo

$$w_2(1+\eta_2) = [w_1(1+\eta_1)]^2(1+\varepsilon_2) = w_1^2(1+\eta_1)^2(1+\varepsilon_2)\,,$$

a kako je $w_2 = w_1^2$, vrijedi

$$\begin{aligned}
\mid \eta_2 \mid &= \mid (1+\eta_1)^2(1+\varepsilon_2) - 1 \mid = \mid \varepsilon_2 + 2\eta_1 + \eta_1^2 + \varepsilon_2(2\eta_1 + \eta_1^2) \mid \\
&\leq \epsilon + 2\epsilon + \epsilon^2 + \epsilon(2\epsilon + \epsilon^2) = 3\epsilon + 3\epsilon^2 + \epsilon^3 = (3 + 3\epsilon + \epsilon^2)\epsilon \\
&\leq 3.00000036\epsilon\,.
\end{aligned}$$

Sljedeća jednadžba

$$z_3 = (1+z_2)(1+\varepsilon_3) = [1+w_2(1+\eta_2)](1+\varepsilon_3) = (1+w_2)(1+\frac{w_2}{1+w_2}\eta_2)(1+\varepsilon_3),$$

zbog $w_2 \geq 0$ *daje*

$$\begin{aligned} \mid \eta_3 \mid &= \mid \frac{w_2}{1+w_2}\eta_2 + \varepsilon_3 + \frac{w_2}{1+w_2}\eta_2\varepsilon_3 \mid \\ &\leq \mid \varepsilon_3 \mid + \mid \eta_2 \mid + \mid \varepsilon_3\eta_2 \mid \leq [1+3.00000036(1+\epsilon)]\epsilon \\ &\leq 4.00000072\epsilon \end{aligned}$$

Pišući

$$Z_4 = \sqrt{z_3}(1+\varepsilon_4) = \sqrt{w_3(1+\eta_3)}(1+\varepsilon_4) = w_4\sqrt{1+\eta_3}(1+\varepsilon_4)\,,$$

i koristeći relaciju (2.5.3), dobivamo

$$\mid \eta_4 \mid \leq \frac{1}{2} \mid \eta_3 \mid + \mid \varepsilon_4 \mid + \frac{1}{2} \mid \eta_3\varepsilon_4 \mid \leq 3.0000005984\epsilon\,.$$

Konačno, $1+\eta_5 = (1+\eta_4)(1+\varepsilon_5)$*, pa je*

$$\mid \eta_5 \mid \leq \mid \eta_4 \mid \; + \mid \varepsilon_5 \mid \; + \mid \eta_4\varepsilon_5 \mid \leq 4.00000096\epsilon\,.$$

Vidimo da korištenjem sofisticiranije formule, koja zahtijeva više strojnih instrukcija od one jednostavne, dobivamo i rezultat koji je generalno nešto netočniji (iako se još uvijek radi o greški u zadnjoj sigurnoj decimali egzaktnog rezulata). Međutim, dobitak je u proširenju domene funkcije koja paru reprezentabilnih brojeva (x,y) *pridružuje* $f\ell(\sqrt{x^2+y^2})$*. To je dosta važno kako bi se izbjeglo prekoračenje i prekid rada računala ili moguća relativna netočnost ako je rezultat mali broj.*

Još smo ostali dužni promotriti što se događa ako su x *ili* y *takvi da* x^2 *ili* y^2 *potkoračuje ili ako* z *postaje nula ili ako* $\sqrt{x^2+y^2}$ *padne u područje subnormalnih brojeva. Ako samo* x^2 (y^2) *potkoračuje u nulu ili subnormalni broj, rezultat će imati malu relativnu grešku, ne veću od nekoliko* ϵ*. Ako izračunati* $f\ell(x^2+y^2)$ *padne u područje subnormalnih brojeva, tada će* $f\ell(x^2+y^2)$ *možda nositi veliku relativnu grešku. Vađenje korijena će tu grešku približno prepoloviti, ali će ona i dalje ostati velika. Potkoraćenje obaju brojeva u nuli će dati maksimalnu relativnu grešku* -1*. Dakle, kad prijeti potkoraćenje ili postepeno potkoraćenje, direktna formula neće dati točan rezultat. Što će dati druga kompliciranija formula?*

Ako je $\mid u \mid \geq N_{min}$*, tada će se* $f\ell(u\sqrt{1+(v/u)^2})$ *izračunati s malom relativnom greškom (kako je gore izračunato) bez obzira na to da li je* v *subnormalni broj ili nije. Ako je pak* u *subnormalni broj, tada su oba polazna broja također subnormalni brojevi i kod smještavanja brojeva u računalo došlo je do gubljenja relativne točnosti. Tada će i egzaktni kvocijent* v/u *i izračunati kvocijent* $f\ell(v/u)$ *imati veću (ili veliku)*

relativnu grešku pa će isto vrijediti i za z. Jedini lijek je da se ulazni podaci, prije učitavanja, x i y skaliraju potencijom od 2 tako da u ne bude subnormalni broj. Nakon računanja treba z skalirati inverznom (negativnom) potencjom broja 2.

U zakjučku možemo reći da kompliciraniju formulu treba koristiti ako u programskom jeziku nije implementirano signaliziranje da je došlo do izuzetne situacije. Ako je signaliziranje implementirano, tada je najbolje koristiti jednostavniji algoritam, uz skretnicu koja u slučaju izuzetne situacije vodi kontrolu na kompliciraniji algoritam. ■

2.5.1. Propagiranje grešaka zaokruživanja

Promotrimo prvo kako jedna računska operacija na računalu proširuje postojeće greške u podacima. Pretpostavit ćemo da su $\hat{x} = x(1+\varepsilon_x)$ i $\hat{y} = y(1+\varepsilon_y)$ podaci u računalu koji aproksimiraju točne podatke x i y s relativnim greškama ε_x i ε_y, respektivno.

Za operaciju množenja vrijedi

$$\begin{aligned} fl(\hat{x}\cdot\hat{y}) &= (\hat{x}\cdot\hat{y})(1+\varepsilon_\times) \qquad\qquad |\varepsilon_\times| \le \epsilon \\ &= xy((1+\varepsilon_x)(1+\varepsilon_y)(1+\varepsilon_\times) \\ &= xy(\varepsilon_x+\varepsilon_y+\varepsilon_x\varepsilon_y+\varepsilon_\times+u), \end{aligned}$$

gdje je $|u| \approx |\varepsilon_\times(\varepsilon_x+\varepsilon_y)| \le \epsilon(|\varepsilon_x|+|\varepsilon_x|)$ po pretpostavci manje od ϵ pa nas dalje ne zanima. Ako su $|\varepsilon_x|$ i $|\varepsilon_y|$ tako mali, tako da je i $|\varepsilon_x\varepsilon_y|$ manje od ϵ i taj član možemo zanemariti. Zaključujemo da relativna greška kod množenja propagira tako da se *zbroju relativnih grešaka faktora pridoda greška nastala kod množenja* kao i umnožak relativnih grešaka u x i y ako je dovoljno velik. Uočimo, tek ako su $|\varepsilon_x|$ i $|\varepsilon_y|$ približno istih modula i suprotnih predznaka, na rezultirajuću grešku može bitnije utjecati $\varepsilon_x\varepsilon_y$.

Za operaciju dijeljenja imamo

$$\begin{aligned} fl(\frac{\hat{x}}{\hat{y}}) &= \frac{\hat{x}}{\hat{y}}(1+\varepsilon_/) = \frac{x}{y}\frac{1+\varepsilon_x}{1+\varepsilon_y}(1+\varepsilon_/), \qquad\qquad |\varepsilon_/| \le \epsilon \\ &= \frac{x}{y}(1+\varepsilon_x)(1-\varepsilon_y+\frac{\varepsilon_y^2}{1+\varepsilon_y})(1+\varepsilon_/) \\ &= \frac{x}{y}(1+\varepsilon_x-\varepsilon_y-\varepsilon_x\varepsilon_y+\frac{\varepsilon_y^2}{1+\varepsilon_y}+\varepsilon_/+u) \end{aligned}$$

gdje ćemo

$$u = (\varepsilon_/ + \frac{\varepsilon_y^2}{1+\varepsilon_y})(\varepsilon_x-\varepsilon_y-\varepsilon_x\varepsilon_y)$$

zanemariti, smatrajući da je manje od osnovne greške zaokruživanja ϵ. Vidimo da je relativna greška dominirana članom $\varepsilon_x - \varepsilon_y - \varepsilon_x\varepsilon_y + \frac{\varepsilon_y^2}{1+\varepsilon_y}$, pri čemu, ako ε_x i ε_y nisu skoro jednaki, možemo zanemariti članove višeg reda $\varepsilon_x\varepsilon_y$ i $\frac{\varepsilon_y^2}{1+\varepsilon_y}$. Kao i kod množenja, nova greška $\varepsilon_/$ ima utjecaj tek na zadnju decimalu binarnog (pogotovo decimalnog) prikaza kvocijenta.

Kod aditivnih operacija imamo

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

Neka su x i y takvi da je $x \pm y = \operatorname{sign}(x)(|x| + |y|)$. To će vrijediti ako je $\pm = +$, $\operatorname{sign}(x) = \operatorname{sign}(y)$ ili $\pm = -$, $\operatorname{sign}(x) = -\operatorname{sign}(y)$. Tada je

$$\begin{aligned} s &= \frac{x}{x \pm y}(\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm) \pm \frac{y}{x \pm y}(\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm) \\ &= \frac{|x|}{|x| + |y|}(\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm) + \frac{|y|}{|x| + |y|}(\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm) \end{aligned} \tag{2.5.4}$$

konveksna suma realnih brojeva $\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm$ i $\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm$, pa je s između tih brojeva. Stoga je

$$\begin{aligned} |s| &\le \max\{|\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm|\,,\ |\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm|\} \\ &\le \max\{|\varepsilon_x|, |\varepsilon_y|\} + |\varepsilon_\pm|(1 + \max\{|\varepsilon_x|, |\varepsilon_y|\}) \\ &\approx \max\{|\varepsilon_x|, |\varepsilon_y|\} + \epsilon\,, \end{aligned} \tag{2.5.5}$$

pa na relativnu grešku rezultata utječu obje "donesene" greške ε_x i ε_y. Finije razmatranje će koristiti relaciju (2.5.4) koja pokazuje da se ε_x množi sa $|x|/(|x| + |y|)$ dok se ε_y množi sa $|y|/(|x| + |y|)$. Ako je npr. $|x| \gg |y|$ i $|\varepsilon_x| \ll |\varepsilon_y|$, tada će $|s|$ biti blizu manje greške $|\varepsilon_x|$.

Kod svih dosadašnjih slučajeva relativna greška u rezultatu nije bitno veća od relativnih grešaka u međurezultatima. Ako polazimo od točnih polaznih podataka, potreban je ogroman broj operacija tih vrsta da bi greška u "zadnjem međurezultatu" bitnije narasla. Međutim još nismo razmatrali preostali slučaj, $x \pm y = \operatorname{sign}(x)(|x| - |y|)$, koji će vrijediti ako je $\pm = +$, $\operatorname{sign}(x) = -\operatorname{sign}(y)$ ili $\pm = -$, $\operatorname{sign}(x) = \operatorname{sign}(y)$. Umjesto relacije (2.5.4), imat ćemo

$$s = \frac{|x|}{|x| - |y|}(\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm) - \frac{|y|}{|x| - |y|}(\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm). \tag{2.5.6}$$

Ako je $|x| \approx |y|$ greške $\varepsilon_x + \varepsilon_\pm + \varepsilon_x\varepsilon_\pm$ i $\varepsilon_y + \varepsilon_\pm + \varepsilon_y\varepsilon_\pm$ će se množiti sa potencijalno vrlo velikim brojevima $|x|/(|x| - |y|)$ odnosno $-|y|/(|x| - |y|)$. Taj fenomen smo već

upoznali kao opasno/katastrofalno "kraćenje". Ako su $|x|/(|x|-|y|)$ i $|y|/(|x|-|y|)$ brojevi reda veličine $|x|$ odnosno $|y|$, onda neće doći do opasnog kraćenja.

U zaključku možemo reći da su operacije množenja i dijeljenja, te zbrajanja sumanada istog predznaka "dobre" operacije koje neće bitnije pogoršati relativnu pogrešku operanada, dok za operaciju oduzimanja brojeva istog predznaka to vrijedi tek ako nije došlo do kraćenja. Ako je došlo do kraćenja, tada vrijedi pravilo: što jače kraćenje u operandima, to veća relativna greška u rezultatu.

Koji puta je kraćenje neizbježno zbog naravi problema (loše uvjetovani problemi), pa možemo tek birati između algoritma koji će dati katastroflno kraćenje u jednoj ili nekoliko operacija, ili algoritam koji će postepeno, jedva primijetljivo, kratiti međurezultate kroz mnogo aditivnih operacija.

Promotrimo sada **akumulaciju** grešaka kod računanja produkta i sume od n brojeva, kao i kod osnovnih vektorskih i matričnih operacija.

2.5.2. Stabilnost produkta od n brojeva

Za produkt brojeva vrijedi

Lema 2.5.2. *Neka je* $x_i = fl(x_i)$, $1 \le i \le n$ *i neka je za svako* $1 \le i \le n$ $|fl(x_1 \cdot x_2 \cdots x_i)| \le N_{max}$. *Tada je*

$$fl(x_1 \cdots x_n) = (x_1 \cdots x_n)(1+\varepsilon), \quad |\varepsilon| \le 1.008(n-1)\epsilon.$$

Dokaz. Ako definiramo pomoćne varijable z_i, sljedeći iteracije algoritma za množenje brojeva $x_1 x_2 \cdots x_n$ u redoslijedu kako je napisano i koristeći relaciju (2.3.4), imat ćemo

$$\begin{aligned} z_1 &= x_1 \\ z_2 &= fl(z_1 \cdot x_2) = (z_1 \cdot x_2)(1+\varepsilon_2) \\ z_3 &= fl(z_2 \cdot x_3) = (z_2 \cdot x_3)(1+\varepsilon_3) \\ &\cdots \\ z_{n-1} &= fl(z_{n-2} x_{n-1}) = (z_{n-2} x_{n-1})(1+\varepsilon_{n-1}) \\ z_n &= fl(z_{n-1} x_n) = (z_{n-1} x_n)(1+\varepsilon_n)\,. \end{aligned}$$

Množeći lijeve i desne strane svih jednakosti, dobit ćemo nakon kraćenja pomoćnih varijabli

$$z = z_n = x_1 \cdots x_n (1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n)\,.$$

Dakle je

$$1+\varepsilon = (1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n)\,, \tag{2.5.7}$$

pa je

$$(1-|\varepsilon_2|)(1-|\varepsilon_3|)\cdots(1-|\varepsilon_n|)-1 \le \varepsilon \le (1+|\varepsilon_2|)(1+|\varepsilon_3|)\cdots(1+|\varepsilon_n|)-1.$$

Kako je

$$1-(1-|\varepsilon_2|)(1-|\varepsilon_3|)\cdots(1-|\varepsilon_n|) \le (1+|\varepsilon_2|)(1+|\varepsilon_3|)\cdots(1+|\varepsilon_n|)-1$$

zaključujemo da je

$$|\varepsilon| \le (1+\epsilon)^{n-1}-1 \le 1.008(n-1)\epsilon\,.$$

Zadnja nejednakost slijedi iz leme 2.5.1.(v) uzimajući $\varepsilon = \epsilon$. ■

Iz dokaza slijedi da ocjena vrijedi za svaki poredak faktora (svaki algoritam koji u nekom poretku množi faktore) ako svi parcijalni produkti ne prekoračuju (tj. za pomoćne varijable vrijedi $z_k \le N_{max}$).

Ovaj rezultat pokazuje da množenje više faktora sporo akumulira relativnu grešku u rezultatu. Ako se koristi način zaokruživanja prema najbližem, greške ε_i iz dokaza će biti i pozitivne i negativne, pa će ukupna greška

$$(1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n)-1 \approx \varepsilon_2+\varepsilon_3+\cdots+\varepsilon_n$$

biti još manja. Npr. umnožak milijun brojeva računan u dvostrukoj točnosti će imati barem 10 točnih značajnih znamenaka, a uz zaokruživanje prema najbližem može se očekivati barem 13 točnih značajnih znamenaka (to će biti jasnije nakon odjeljka o sumiranju.

Formalno, lema 2.5.2. pokazuje da je svaki algoritam koji uzastopce množi brojeve stabilan unaprijed. Iz dokaza leme vidimo da rezultat možemo zapisati u obliku

$$f\ell(x_1\cdots x_n) = x_1[x_2(1+\varepsilon_2)][x_3(1+\varepsilon_3)]\cdots[x_n(1+\varepsilon_n)], \quad |\varepsilon_i| \le \epsilon,\ 2 \le i \le n\,.$$

Ako interpretiramo faktore $x_i(1+\varepsilon_i)$ kao malo perturbirane polazne podatke x_i, vidimo da je algoritam za produkt od n brojeva savršeno stabilan unazad. Izračunati produkt je jednako točan kao kad polazne brojeve koji nisu BPZ učitamo u računalo i zatim ih pomnožimo egzaktno!

Iz relacije (2.5.7) slijedi

$$\begin{aligned}\varepsilon &= (1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n)-1\\ &= \varepsilon_2+\varepsilon_3+\cdots+\varepsilon_n+\sum_{2\le i<k\le n}\varepsilon_i\varepsilon_k+\cdots+\varepsilon_2\varepsilon_3\cdots\varepsilon_n\\ &= \varepsilon_2+\varepsilon_3+\cdots+\varepsilon_n+\mathcal{O}(\epsilon^2)\end{aligned}$$

pa se iskaz leme još zapisuje u obliku

$$fl(x_1 \cdots x_n) = (x_1 \cdots x_n)(1+\varepsilon), \quad |\varepsilon| \leq (n-1)\epsilon + \mathcal{O}(\epsilon^2). \tag{2.5.8}$$

Prednost takvog zapisa je u tome što daje najmanju moguću konstantu uz linerni član ϵ. Međutim, ako se taj rezultat koristi kao dio analize nekog složenijeg izraza ili algoritma postoje dvije neugodnosti. Prvo, ako u nejednakosti postoji član oblika $\mathcal{O}(\epsilon^2)$, više nemamo čistu ocjenu već tek ocjenu "do na veličinu reda ϵ^2". Drugo, ako se u daljoj analazi član $\mathcal{O}(\epsilon^2)$ množi sa potencijlno velikim brojevima, više nismo sigurni je li postao $\mathcal{O}(\epsilon)$ ili možda čak $\mathcal{O}(1)$, pa je svrha analize narušena. Stoga se oblik (2.5.8) uglavnom koristi kod jednostavnijih izraza u svrhu dobivanja što povoljnije konstante uz ϵ.

Jedan korisni pomočni rezultat

U formuli (2.5.7) se javlja produkt faktora koji su oblika $1+\varepsilon_k$. Za ocjenjivanje izraza koji sadrže samo množenja i dijeljenja možemo koristiti sljedeću lemu.

Lema 2.5.3. *Neka je $u > 0$ realni broj i n cijeli pozitivni broj takav da vrijedi $nu < 1$. Neka su za $i = 1, \ldots, n$, δ_i realni brojevi, a $p_i \in \{-1, 1\}$. Ako su svi $|\delta_i| \leq u$, onda vrijedi*

$$\prod_{i=1}^{n} (1+\delta_i)^{p_i} = 1 + \theta_n, \tag{2.5.9}$$

uz ocjenu

$$|\theta_n| \leq \gamma_n := \frac{nu}{1-nu}. \tag{2.5.10}$$

Dokaz. Dokaz se provodi indukcijom po n. Za $n = 1$ vrijedi $u < 1$. Ako je $p_1 = 1$, onda je $\theta_1 = \delta_1$, pa je $|\theta_1| \leq u \leq u/(1-u)$. Ako je $p_1 = -1$, onda je $1 + \theta_1 = 1/(1+\delta_1)$, pa je

$$\theta_1 = \frac{1}{1+\delta_1} - 1 = -\frac{\delta_1}{1+\delta_1}.$$

S obzirom da je $1 + \delta_1 \geq 1 - u > 0$, dobivamo

$$|\theta_1| = \frac{|\delta_1|}{1+\delta_1} \leq \frac{u}{1-u},$$

čime je lema dokazana za $n = 1$.

Pretpostavimo da tvrdnja leme vrijedi za neko $n \geq 1$. To znači da je $nu < 1$, pa je $1 + \delta_i > 0$ za sve $1 \leq i \leq n$. Dakle je produkt na lijevoj strani od (2.5.9) pozitivan, pa je θ_n dobro definiran i zadovoljava ocjenu (2.5.10).

Pokažimo da tvrdnja leme vrijedi za $n+1$. Iz pretpostavke $(n+1)u < 1$ i $|\delta_i| \leq u$, $1 \leq i \leq n+1$ slijedi $nu < 1$ i $|\delta_i| \leq u$, $1 \leq i \leq n$ pa možemo koristiti

relaciju (2.5.9) i ocjenu (2.5.10). Također, broj θ_{n+1} je dobro definiran relacijom $1+\theta_{n+1} = \prod_{i=1}^{n+1}(1+\delta_i)^{p_i}$, pa je

$$1+\theta_{n+1} = \prod_{i=1}^{n}(1+\delta_i)^{p_i}\cdot(1+\delta_{n+1})^{p_{n+1}} = (1+\theta_n)(1+\delta_{n+1})^{p_{n+1}}\,.$$

Za $p_{n+1}=1$ dobivamo

$$\theta_{n+1} = \theta_n + \delta_{n+1} + \theta_n\delta_{n+1}$$

pa je

$$|\theta_{n+1}| \le \frac{nu}{1-nu} + u + \frac{nu^2}{1-nu} = \frac{(n+1)u}{1-nu} < \frac{(n+1)u}{1-(n+1)u},\,.$$

Za $p_{n+1}=-1$ dobivamo

$$\theta_{n+1} = \frac{1+\theta_n}{1+\delta_{n+1}} - 1 = \frac{\theta_n - \delta_{n+1}}{1+\delta_{n+1}},$$

pa je

$$|\theta_{n+1}| \le \frac{|\theta_n| + |\delta_{n+1}|}{1+\delta_{n+1}}.$$

Kako je $1+\delta_{n+1} \ge 1-u > 0$, lako slijedi

$$|\theta_{n+1}| \le \frac{\dfrac{nu}{1-nu}+u}{1-u} = \frac{(n+1)u-nu^2}{1-(n+1)u+nu^2} < \frac{(n+1)u}{1-(n+1)u}\,.$$

Time je dokaz indukcijom završen. ■

Brojevi γ_n, $n \ge 1$ imaju za primjene vrlo praktično svojstvo,

$$\gamma_m + \gamma_n + \gamma_m\gamma_n \le \gamma_{m+n}\,. \tag{2.5.11}$$

Zadatak 2.5.1. *(i) Dokažite nejednakost (2.5.11).*
(ii) Primijenite lemu 2.5.3. na relaciju (2.5.7). Je li dobivena konstanta uz ϵ veća ili manja od 1.008? ■

Nakon analize stabilnosti produkta brojeva, proučimo stabilnost sume brojeva.

2.5.3. Stabilnost sume

Neka je

$$s_n = x_1 + x_2 + \cdots + x_n,$$

pri čemu za brojeve x_i vrijedi kao i prije $x_i = f\ell(x_i)$, $1 \le i \le n$. Suma s_n se može računati na razne načine. Promotrimo prvo najjednostvniji *rekurzivni algoritam*

$s_1 := x_1;$
for $i := 2$ **to** n **do**
  $s_i := s_{i-1} + x_i;$

Označimo sa $\hat{s}_i$ izračunatu vrijednost s_i. Tada prema relaciji (2.3.4) vrijedi

$$\hat{s}_2 = f\ell(x_1 + x_2) = (x_1 + x_2)(1 + \varepsilon_1) = x_1(1 + \varepsilon_1) + x_2(1 + \varepsilon_1), \quad |\varepsilon_1| \leq \epsilon\,.$$

Slično,

$$\begin{aligned}\hat{s}_3 &= f\ell(\hat{s}_2 + x_3) = (\hat{s}_2 + x_3)(1 + \varepsilon_2)\\ &= x_1(1 + \varepsilon_1)(1 + \varepsilon_2) + x_2(1 + \varepsilon_1)(1 + \varepsilon_2) + x_3(1 + \varepsilon_2)\,.\end{aligned}$$

Nastavljajući na taj način, dobivamo

$$\begin{aligned}\hat{s}_n &= f\ell(\hat{s}_{n-1} + x_n) = (\hat{s}_{n-1} + x_n)(1 + \varepsilon_{n-1})\\ &= x_1(1 + \varepsilon_1)(1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ &\quad + x_2(1 + \varepsilon_1)(1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ &\quad + x_3(1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ &\quad \cdots\\ &\quad + x_{n-1}(1 + \varepsilon_{n-2})(1 + \varepsilon_{n-1})\\ &\quad + x_n(1 + \varepsilon_{n-1})\,,\end{aligned}$$

gdje su svi $|\varepsilon_i| \leq \epsilon$. Zadnju relaciju možemo pojednostaviti uvođenjem novih varijabli

$$\begin{aligned}1 + \eta_1 &= (1 + \varepsilon_1)(1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ 1 + \eta_2 &= (1 + \varepsilon_1)(1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ 1 + \eta_1 &= (1 + \varepsilon_2)\cdots(1 + \varepsilon_{n-1})\\ &\cdots\\ 1 + \eta_{n-1} &= (1 + \varepsilon_{n-2})(1 + \varepsilon_{n-1})\\ 1 + \eta_n &= 1 + \varepsilon_{n-1}\,.\end{aligned}$$

Tada prethodnu relaciju možemo zapisati u obliku

$$\hat{s}_n = x_1(1 + \eta_1) + x_2(1 + \eta_2) + \cdots x_n(1 + \eta_n) \tag{2.5.12}$$

pri čemu η_i možemo ocijeniti pomoću leme 2.5.3. ili još oštrije pomoću leme 2.5.1.(v),

$$|\eta_i| \leq 1.008 \cdot \begin{cases}(n-1)\epsilon\,, & i = 1\\ (n-i+1)\epsilon\,, & 2 \leq i \leq n\end{cases}. \tag{2.5.13}$$

Relacije (2.5.12) i (2.5.13) pokazuju da je rekurzivni algoritam za sumiranje brojeva povratno stabilan. Ocjene za relativnu grešku svakog perturbiranog polaznog podatka $x_i(1 + \eta_i)$ su manje od $n\epsilon$. Da li je algoritam također stabilan unaprijed?

Da bi odgovorili na to pitanje oduzmimo od $\hat{s}_n$ egzaktnu sumu s_n,

$$\hat{s}_n - s_n = x_1\eta_1 + x_2\eta_2 + \cdots x_n\eta_n$$

i ocijenimo apsolutnu vrijednost te razlike

$$\begin{aligned} |\hat{s}_n - s_n| &\leq |x_1|\,|\eta_1| + |x_2|\,|\eta_2| + \cdots |x_n|\,|\eta_n| \\ &\leq \max_i\{|\eta_i|\}(|x_1| + |x_2| + \cdots + |x_n|) \\ &\leq (|x_1| + |x_2| + \cdots + |x_n|)\cdot 1.008(n-1)\epsilon\,. \end{aligned} \tag{2.5.14}$$

Stoga je relativna greška od $\hat{s}_n$ kao aproksimacije od s_n ocijenjena s

$$\frac{|\hat{s}_n - s_n|}{|s_n|} \leq \mathrm{cond}(s_n)1.008(n-1)\epsilon, \tag{2.5.15}$$

gdje je

$$\mathrm{cond}(s_n) = \frac{|x_1| + |x_2| + \cdots + |x_n|}{|x_1 + x_2 + \cdots + x_n|}\,. \tag{2.5.16}$$

Ako načas pretpostavimo da je $\eta_i x_i$ proizvoljna perturbacija od x_i za svako i, onda vidimo da formula (2.5.14) daje oštru ocjenu jer se dostiže npr. za pozitivne x_i i $\eta_1 = \eta_2 = \cdots = \eta_n$. Nejednakost (2.5.14) pokazuje kako se ponaša relativna promjena funkcije s_n pri relativnim perturbacijama njenih argumenata x_i. Stoga je $\mathrm{cond}(s_n)$ broj uvjetovanosti za s_n i on, kao što znamo iz odjeljka o uvjetovanosti, povezuje grešku unaprijed s greškom unazad. Kako $\mathrm{cond}(s_n)$ može biti proizvoljno veliko (uzmimo npr. brojeve $1,-1$ i t, pa pustite da $t \to 0$) algoritam nije općenito stabilan unaprijed. Ipak, *ako svi x_i imaju isti predznak*, onda je $\mathrm{cond}(s_n) = 1$, pa je i greška unaprijed mala, *a to znači da je algoritam stabilan unaprijed*. Isti zaključak vrijedi kadgod $\mathrm{cond}(s_n)$ nije veliki broj.

Umjesto da sumiranje vršimo po redu od x_1 do x_n, možemo koristiti bilo koji redoslijed. Ocjene će nakon odgovarajuće analize grešaka zaokruživanja biti posve analogne. Pritom će vrijediti relacija (2.5.12) s time da umjesto x_1 i x_2 stoje oni sumandi koji se prvi zbrajaju, umjesto x_3 onaj sumand koi se sljedeći dodaje sumi itd.

Kadkad su sve komponente istog predznaka i pritom još uređene u monotoni niz. Tada se postavlja pitanje u kojem redoslijedu treba sumirati da bi relativna greška u $\hat{s}_n$ bila što manja? Prvo uočimo da je traženje najmanje relativne greške za taj problem ekvivalentno traženju najmanje apsolutne greške (jer nju dijelimo sa s_n, a s_n ne ovisi o redoslijedu sumiranja). Dakle, tražimo redoslijed indeksa j_1, $j_2,\ldots,j_n$ za koji je $|x_{j_1}\eta_1 + x_{j_2}\eta_2 + \cdots + x_{j_n}\eta_n|$ najmanje. Problem je težak jer greške ε_k koje definiraju svaki η_i mogu biti i pozitivne i negativne, pa $|\eta_1|$ može biti manji od bilo kojeg η_j, $3 \leq j \leq n$. Stoga u moramo zahtjev malo oslabiti pitanjem *koji redoslijed indeksa j_1, $j_2,\ldots,j_n$ daje najbolju ocjenu za* $|x_{j_1}\eta_1 + \cdots + x_{j_n}\eta_n|$? Tako dolazimo do jednostavnog pravila:

ako je niz $|x_1|$, $|x_2|$, ..., $|x_n|$ *monoton, sumirajmo u smjeru od apsolutno najmanjeg do apsolutno najvećeg člana.*

Kaskadno sumiranje

Osim danog jednostavnog algoritma za sumiranje spomenimo još jedan koji daje najmanju ocjenu za grešku, ali na računalu traži cijelo polje dodatnih lokacija u memoriji. Možemo ga zvati *sumiranje po parovima* ili *kaskadno sumiranje*. Algoritam ima približno $\log_2(n)$ glavnih koraka, a približno tolika će biti i ograda za *svaki* $|\eta_i|$ iz relacije (2.5.12). Ako je $n = 2^k$, $k \geq 2$ algoritam ima k glavnih koraka. U prvom zbraja po parovima $z_i = x_{2i-1} + x_{2i}$, $1 \leq i \leq [n/2]$

$$z_1 = x_1 + x_2,\ z_2 = x_3 + x_4, \ldots, z_{m_1} = x_{n-1} + x_n, \qquad m_1 = n/2 = 2^{k-1}.$$

U drugom ponavlja istu operaciju nad $z_1, \ldots, z_{m_1}$,

$$w_1 = z_1 + z_2,\ w_2 = z_3 + z_4, \ldots, w_{m_2} = z_{m_1-1} + z_{m_1}, \qquad m_2 = m_1/2 = 2^{k-2}.$$

Nastavljajući na taj način, nakon $k-1$ glavnih koraka imamo $2^{(k-(k-1))} = 2$ člana, pa se u k-tom koraku oni zbroje dajući s_n.

Zadatak 2.5.2. *Ako je* $n = 2^k$*, dokažite da svaki* x_i *sudjeluje u točno* k *zbrajanja, pa je u relaciji (2.5.12) svaki* $|\eta_i|$ *omeđen sa* $1.008k\epsilon$*. Pokušajte napisati algoritam za opći slučaj kada* n *nije potencija od* 2*. Koliko prolaza ima glavna petlja kad je* n *jednako* 8, 16, 9, 10, 11, 12, 13, 14, 15*? Generalizirajte zaljučak za* n *između* 2^K *i* 2^{k+1}*. Koliko zbrajanja prolazi svaki* x_i*? Koliki je ukupni broj računskih operacija? Koliko dodatne memorije (dodatnih ćelija za pomoćne varijable) bi vi koristili za realizaciju algoritma, ako želite sačuvati vrijednosti polaznog niza u polaznom polju?* ■

Jedna generalnija analiza sumacijskih metoda i pripadnih grešaka zaokruživanja (vidi [2]) daje sljedeću generalnu uputu:

Kod odabiranja sumacijske metode s ciljem dobivanja što točnije sume s_n*, treba izabrati onu koja daje što manje apsolutne vrijednosti međurezultata.*

Zadatak 2.5.3. *Prije primjene zadnjeg algoritma možemo sortirati elemente niza* (x_i)*. Koju strategiju biste izabrali? Npr.* $y_1 = x_{min} + x_{max}$ *i kako dalje? ili* $y_1 = x_{min} + x_{next\ min}$ *i kako dalje?* ■

Napomena 2.5.1. *Prednost standardnog algoritma prema "binarnom" je i u kraćem trajanju na računalu. Ne radi se tu o većem broju zbrajanja nego o načinu dohvaćanje elemenata polja u računalu kod izvođenja aritmetičkih operacija s elementima polja. Prije zbrajanja operanada moderna računala zahvate cijeli blok*

podataka (onih na uzastopnim ćelijama) oko tekućeih operanada u brzu cache memoriju. Ako je svih elemenata x_i previše da bi svi stali u brzu memoriju, dohvaćaju se po blokovima. Ako se binarni algoritam ne programira vrlo precizno uz jasno poznavanje kako koristi brzu memoriju, mogao bi zahtijevati daleko veći broj dohvaćanja članova niza x_i nego standardni algoritam. ■

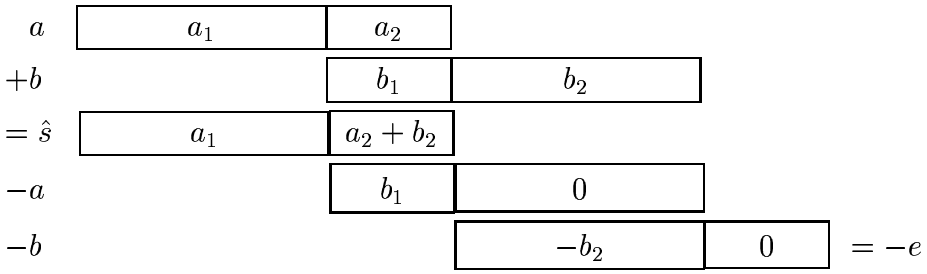
Da li se isplati investirati dodatnu memoriju i računsko vrijeme da bi imali osigurane bitno manje ograde za svaki η_i nego kod standardnog algoritma? Kod većine aplikacija ne. Usprkos mnogo većim ogradama stvarne greške η_i kod standardnog algoritma su bitno manje.

Malo više svjetla na tu zagonetku baca i statistička analiza za sumu $\varepsilon_1 + \cdots + \varepsilon_n$ osnovnih grešaka zaokruživanja. Analiza koja pretpostavlja da su osnovne greške nezavisne slučajne varijable sa medijanom nula i konačnom standardnom devijacijom pokazuje da je očekivana vrijednost za njihovu sumu omeđena sa $\sqrt{n}\epsilon$ što je daleko povoljnije od teoretske granice $n\epsilon$ (npr. očekivana vrijednost za η_1 je oko $\sqrt{n-1}\epsilon$, što je mnogo povoljnije od $(n-1)\epsilon$).

2.5.4. Kompenzirano sumiranje

Kompenzirano sumiranje je posebno atraktivno ako već koristimo najpreciznije aritmetiku u računalu (npr. dvostruku preciznost podataka i aritmetike). Radi se o rekurzivnom algoritmu s korektivnim sumandom koji je tako određen da gotovo poništi greške zaokruživanja.

Neka su $a = f\ell(a)$ i $b = f\ell(b)$ takvi da je $|a| \geq |b|$. Neka je $\hat{s} = f\ell(a + b)$ i promotrimo sljedeću ilustraciju koja koristi izdužene pravokutnike za prikaz mantise. Pri zbrajanju na akumulatoru (aritmetičkoj jedinici procesora) mantisa manjeg po modulu broja se prvo pomakne u desno za broj bitova koji odgovara razlici eksponenata kod brojeva a i b. Zatim se mantise zbroje, zaokruže na t bitova (koliko mantisa dopušta), a ako je pritom došlo do prekoračenja vrijednosti za mantisu one se normalizira, a eksponent podesi. Iz slike 2.5.4. možemo zaključiti



Slika 2.5.1. Izvlačenje greške zaokruživanja

da je greška e,

$$e = -[((a+b)-a)-b] = (a-\hat{s})+b,$$

ako se izračuna na stroju u redoslijedu naznačenom zagradama (označimo ju sa $\hat{e} = ((a \oplus b) \ominus a) \ominus b)$) dobra aproksimacija greške $(a+b) - \hat{s}$. Zapravo za standardni način zaokruživanja u IEEE aritmetici može se pokazati $a+b = \hat{s}+\hat{e}$ pa $\hat{e}$ reprezentira pravu grešku.

Sljedeći algoritam od Kahana koristi korekciju e u svakom koraku standardnog rekurzivnog sumacijskog algoritma. Nakon što je izračunata parcijalna suma s_{i-1}, odmah se računa njena korekcija, koja se u sljedećem koraku dodaje članu x_i prije njegova dodavanja parcijalnoj sumi s_{i-1}. Gotovo uvijek je parcijalna suma po modulu veća od elementa koji joj se dodaje, pa korekcija e koja je mala može mijenjati element x_i, ali ne može mijenjati s_{i-1} koja je već najbolja strojna aproksimacija od $s_{i-2} + x_{i-1}$. Evo algoritma

$s := 0$; $e := 0$;
for $i := 1$ **to** n **do**
begin
  $temp := s$;
  $z := x_i + e$;
  $s := temp + z$;
  $e := (temp - s) + z$ {važan je redoslijed izvršavanja}.
end

Ova metoda ipak ima dva manja nedostatka. Prvo, $\hat{e}$ nije nužno točna korekcija jer uvjet $|s_{i-1}| \geq |x_i|$ (u algoritmu: $|temp| \geq |y|$) neće uvijek biti zadovoljen. Drugo, u računalu se računa $z := x_i \oplus e$ a ne $z := x_i + e$. Ipak se može pokazati da η_i iz formule (2.5.12) za kompenziranu sumaciju zadovoljava $|\eta_i| \leq 2\epsilon + \mathcal{O}(n\epsilon^2)$ što je gotovo nedostižan rezultat.

2.5.5. Stabilnost skalarnog produkta i osnovnih matričnih operacija

Neka su $x, y \in \mathbf{R}^n$, $x = [x_1, \ldots, x_n]^T$, $y = [y_1, \ldots, y_n]^T$. Da bi analizirali skalarni produkt vektora x i y, označimo $s = s_n = x^T y = x_1y_1 + x_2y_2 + \cdots + x_ny_n$ te za svako i parcijale sume $s_i = x_1y_1 + \cdots + x_iy_i$. Kao i kod sumacije, možemo definirati više algoritama za skalarni produkt, ali ćemo zbog jednostavnosti analizirati samo rekurzivni algoritam

$s_1 := x_1y_1$;
for $i := 2$ **to** n **do**
  $s_i := s_{i-1} + x_iy_i$;

Zbog sličnosti sa prijašnjim algoritmom sa sumiranje n brojeva, možemo preuzeti neke dijelove izvoda za ocjenu grešaka zaokruživanja. Kao i prije, sa $\hat{s}_i$ označimo

izračunate vrijednosti. Tada imamo,

$$\begin{aligned}
\hat{s}_1 &= f\ell(x_1y_1) = x_1y_1(1+\delta_1), \qquad |\delta_1| \le \epsilon \\
\hat{s}_2 &= f\ell(\hat{s}_1 + x_2y_2) = [\hat{s}_1 + x_2y_2(1+\delta_2)]\,(1+\varepsilon_1) \\
&= x_1y_1(1+\delta_1)(1+\varepsilon_1) + x_2y_2(1+\delta_2)(1+\varepsilon_1) \\
\hat{s}_3 &= f\ell(\hat{s}_2 + x_3y_3) = [\hat{s}_2 + x_3y_3(1+\delta_3)]\,(1+\varepsilon_2) \\
&= x_1y_1(1+\delta_1)(1+\varepsilon_1)(1+\varepsilon_2) + x_2y_2(1+\delta_2)(1+\varepsilon_1)(1+\varepsilon_2) \\
&\quad + x_3y_3(1+\delta_3)(1+\varepsilon_2) \\
&\quad \cdot \quad \cdot \quad \cdot \\
\hat{s}_n &= f\ell(\hat{s}_{n-1} + x_ny_n) = [\hat{s}_{n-1} + x_ny_n(1+\delta_n)]\,(1+\varepsilon_{n-1}) \\
&= x_1(1+\delta_1)(1+\varepsilon_1)(1+\varepsilon_2)\cdots(1+\varepsilon_{n-1}) \\
&\quad + x_2(1+\delta_2)(1+\varepsilon_1)(1+\varepsilon_2)\cdots(1+\varepsilon_{n-1}) \\
&\quad + x_3(1+\delta_3)(1+\varepsilon_2)\cdots(1+\varepsilon_{n-1}) \\
&\quad \cdot \ \cdot \ \cdot \\
&\quad + x_{n-1}(1+\delta_{n-1})(1+\varepsilon_{n-2})(1+\varepsilon_{n-1}) \\
&\quad + x_n(1+\delta_n)(1+\varepsilon_{n-1})\,,
\end{aligned}$$

pa je

$$\hat{s}_n = x_1y_1(1+\eta_1) + x_2y_2(1+\eta_2) + \cdots x_ny_n(1+\eta_n), \tag{2.5.17}$$

pri čemu η_i možemo ocijeniti pomoću leme 2.5.3. ili još oštrije pomoću leme 2.5.1.(v),

$$|\eta_i| \le 1.008 \cdot \begin{cases} n\epsilon\,, & i = 1 \\ (n-i+2)\epsilon\,, & 2 \le i \le n \end{cases}. \tag{2.5.18}$$

Relacija (2.5.17) je rezultat analize povratne greške koji se može ovako interpretirati:

Izračunati skalarni produkt $f\ell(x^Ty)$ je egzaktni skalarni produkt malo perturbiranih vektora x i y. Algoritam je povratno stabilan..

Kao perturbirane vektore možemo uzeti npr. $x+\delta x = [x_1(1+\eta_1), \ldots, x_n(1+\eta_n)]$ i y, također x i $y+\delta y = [y_1(1+\eta_1), \ldots, y_n(1+\eta_n)]$, ali i

$$x+\delta x = [x_1\sqrt{1+\eta_1}, \ldots, x_n\sqrt{1+\eta_n}] \quad \text{i} \quad y = [y_1\sqrt{1+\eta_1}, \ldots, y_n\sqrt{1+\eta_n}].$$

Da bi dobili ocjenu za grešku unaprijed, koristimo analogne ocjene kao u relaciji (2.5.14),

$$\begin{aligned}
\frac{|\hat{s}_n - s_n|}{|s_n|} &\le \frac{|x_1y_1| + |x_2y_2| + \cdots + |x_ny_n|}{|x_1y_1 + x_2y_2 + \cdots + x_ny_n|} \max_i\{|\eta_i|\} \\
&\le \mathrm{cond}(s_n) 1.008 n\epsilon\,.,
\end{aligned} \tag{2.5.19}$$

Vidimo da broj uvjetovanosti $\mathrm{cond}(s_n)$ za skalarni produkt može biti po volji velik. To se događa kad je $|x^Ty| \ll |x|^T|y|$. Zaključak je isti kao i kod sumiranja: greška

unaprijed može biti velika, pa algoritam nije stabilan unaprijed. Ipak on je stabilan unaprijed kad su svi produkti $x_i y_i$ istog predznaka (tada je $\mathrm{cond}(s_n) = 1$).

Jedan specijalni slučaj, računanje norme $\|x\| = \sqrt{x_1^2 + \cdots + x_n^2}$ zaslužuje razmatranje. S obzirom da je za taj slučaj broj uvjetovanosti 1, vrijedi

$$f\ell(x_1^2 + \cdots + x_n^2) = (x_1^2 + \cdots + x_n^2)(1 + \varepsilon), \qquad |\varepsilon| \leq 1.008n\epsilon\,,$$

pa se jednostavno dobije

$$f\ell(\|x\|) = \|x\|(1 + \varepsilon_{\|\cdot\|}), \quad |\varepsilon_{\|\cdot\|}| \leq \sqrt{1 + 1.008n\epsilon}(1 + \epsilon) - 1 \leq (1 + 0.504n)\epsilon$$

što je manje od $0.504(n+1)\epsilon$ za $n \geq 2$. Dakle računanje norme je stabilno (unaprijed i unazad).

Zadatak 2.5.4. *Da li je računanje ostalih normi, npr.* $\|x\|_1$, $\|x\|_\infty$, $\|x\|_p$, *stabilno?* ■

Za razliku od unutrašnjeg produkta, vanjski produkt vektora $A = xy^T$ nije povratno stabilan. Ovdje čak x i y ne trebaju biti iste dimenzije. Ako elemente od A označimo s a_{ij}, onda vrijedi $a_{ij} = x_i x_j$, pa je $\hat{a}_{ij} = f\ell(a_{ij}) = a_{ij}(1 + \varepsilon_{ij})$, $|\varepsilon_{ij}| \leq \epsilon$ za sve i, j. Stoga je $\hat{A} = f\ell(xy^T) = A + E$, $E = (\varepsilon_{ij})$. Stoga vrijedi $|E| \leq \epsilon|xy^T| = \epsilon|A|$ odnosno $|\hat{A} - A| \leq \epsilon|A|$ pa je računanje vanjskog produkta stabilna unaprijed operacija. Rzlika između tipova stabilnosti unutrašnjeg i vanjskog produkta izražava generalni princip: numerički proces će prije biti stabilan unazad (unaprijed) ako ima više (manje) ulaznih nego izlaznih podataka.

Literatura

[1] D. Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic. ACM Computing Surveys, Vol. 23, No. 1, March 1991.

[2] N. J. Higham: Accuracy and Stability of Numerical algoritams. SIAM, Philadelpia 1996.

[3] M. L. Overton: Numerical Computing with IEEE Floating Point Arithmetic. SIAM, Philadelpia 2001.

[4] J. H. Wilkinson: Rounding Errors in Algebraic Processes. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New-York, 1994, ISBN 0-486-67999-5.