1 Arytmetyka IEEE

W standarcie IEEE liczby tak zwanej podwójnej precyzji reprezentowane są przez 64 bity, z czego jeden bit to znak, 11 to wykładnik a 52 bity to mantysa. Matematycznie liczby IEEE to skończony podzbiór zbioru liczb wymiernych składający się z liczb postaci

$$sm2^e$$

gdzie s to znak, m to przeskalowana mantysa zaś e to przesunięty wykładnik. Dla normalnych liczb IEEE można przyjąc że m jest liczbą całkowitą z przedziału $[2^{52}, 2^{53} - 1]$, zaś e jest liczbą całkowitą z przedziału $[-1075, 970]$. Niesymetria wykładników jest spowodowana tym że u nas mantysa jest liczbą całkowitą. Zwykle przyjmuje się że mantysa leży w przedziale $[1, 2)$ i wtedy wykładnik byłby z przedziału $[-1022, 1023]$. Zakres m jest ograniczony z dołu przez dużą liczbę i dzięki temu nie trzeba pamiętać najwyższego bitu m, czyli do pamiętania m faktycznie wystarczą 52 bity (często pisze się że mantysa zawiera ukryty bit).

Wykładnik pamiętany jest jako liczba dwójkowa bez znaku, odejmując od niej odpowiednią stałą (u nas 1076, w standartowym opisie 1023) dostaje się właściwą wartość. Dwie wartości wykładnika używa się do specjalnych celów.

Oprócz normalnych liczb IEEE przewiduje też liczby nieznormalizowane podobnej postaci jak wyżej, tyle że wykładnik to wzorzec bitowy złożony z samych zer, zaś m jest liczbą całkowitą z przedziału $[1, 2^{52} - 1]$. Liczby nieznormalizowane zostały wprowadzone po to by w przypadku gdy w obliczeniach pojawią się małe wykładniki utrata dokładności następowała stopniowo. Jeśli potrzebujemy pełnej dokładności to liczby nieznormalizowane są szkodliwe, z ich powodu możemy dostać mniejszą dokładność obliczeń niż oczekiwana.

Jeśli wszystkie pozycje bitowe liczby oprócz znaku są zerowe to liczbę traktuje się jako zero. Oznacza to że zero ma znak. Standard podaje dokładne reguły jaki będzie znak zera z różnych operacji, jednakże lepiej to ignorować. W normalnych sytuacjach znak zera nie jest istotny.

Standard IEEE przewiduje też inne wielkości niż liczby. W szczególności wprowdza plus i minus nieskończoność i "nie-liczby" (ang. NAN). Procesory generują NAN wtedy gdy pracują w trybie kontynuacji obliczeń po błędach i nie ma sensowego wyniku. Np. odjęcie od siebie dwu nieskończoności daje NAN.

Dla nas różne dziwne wielkości ze standartu IEEE są niepotrzebne, tyle że ich pojawienie się obliczeniach faktycznie oznacza błąd. Dla prawdziwych liczb oficjalne reguły arytmetyki IEEE są bardzo proste: operacje wykonuje się dokładnie a następnie wynik zaokrągla się do reprezentowalnej wartości. Jeśli wynik ma zbyt duża wartość bezwzględną (jest poza zakresem reprezentowalnych liczb), oznacza to tzw. przepełnienie. Zależnie od typu konkretnego procesora i różnych ustawień przepełnienie może spowodować przerwanie obliczeń. Jeśli obliczenia są kontynuowane to jako wynik produkuje się odpowiednią nieskończoność. Podobnie zależnie od ustawień procesora jeśli wynik jest zbyt mały co do wartości bezwzględnej (lecz nie jest dokładnie zerem) to może być zastą-

piony przez zero, przez liczbę nieznormalizowaną czy spowodować przerwanie obliczeń.

Warto tu zauważyć że w praktyce procesory nie stosują literalnie podanej reguły, okazuje się że wystarczy odpowiednio obliczyć kilka (w zasadzie 2) dodatkowych bitów ponad n bitową reprezentację by móc podać poprawnie zaokrąglony wynik.

IEEE podaje kilka możliwych trybów zaokrąglania. Najważniejszy jest tryb zaokrąglania do najbliższej reprezentowalnej wartości, z dodatkową regułą mówiącą że jeśli liczba jest równo odległa od dwu liczb reprezentowalnych to wybieramy tak by najmiej znacząca cyfra dwójkowa była parzysta. Inne zdefinowane tryby zaokrągleń to zaokrąglanie zawsze w dół, zaokrąglanie zawsze w górę i zaokrąglanie w kierunku zera. Tryby zaokrąglania w dół i zaokrąglania w górę mogą pomóc przy implementacji arytmetyki przedziałowej. Tryb zaokrąglania w kierunku zera pozwala zaimplementować wyspecyfikową w języku C regułę konwersji wartości zmiennopozycyjnych do całkowitych. Poza takimi dość specjalnymi użyciami specjalne tryby zaokrąglania są mało przydatne.

1.1 Błędy zaokrąglenia w artymetyce IEEE

Zakładając że wynik jest liczbą normalną różną od zera, tzn. nie było przepełninia i nie pojawiły się liczby nieznormalizowane i że używamy zaokrąglanie do najbliższej reprezentowalnej wartości to błąd operacji łatwo oszacować. Mianowicie, jeśli e_{obl} to obliczony wynik zaś e_{dok} to dokładny wynik operacji to mamy

$$|e_{obl} - e_{dok}| \leq \varepsilon |e_{obl}|$$

gdzie $\varepsilon = 2^{-53}$. Jeśli chcemy szacować błąd przez dokładną wartość wyniku to mamy bardzo podobne oszacowanie

$$|e_{obl} - e_{dok}| \leq \varepsilon |e_{dok}|.$$

Oba oszacowania uzasadniamy w podobny sposób: różnice między kolejnymi liczbami reprezentowalnymi są stałe w przedziałach postaci $[2^k, 2^{k+1}]$ i wynoszą $2^{-52}2^k$. A więc zaokrąglanie wprowadzi maksymalnie bład $2^{-53}2^k = \varepsilon 2^k$. Jeśli $e_{dok} \in [2^k, 2^{k+1}]$ to również $e_{obl} \in [2^k, 2^{k+1}]$ i w szczególności $|e_{obl}| \geq 2^k$ oraz

$$|e_{obl} - e_{dok}| \leq \varepsilon 2^k \leq \varepsilon |e_{obl}|.$$

Podobny argument działa dla liczb ujemnych do daje pierwsze oszacowanie. Dowód drugiengo oszacowanie jest podobny.

1.2 Pojedyncza precyzja

Ogólne własności arytmetyki pojedynczej precyzji są podobne jak arytmetyki podwójnej precyzji. Różnią się specyficzne parametry: reprezentacja jest 32 bitowa, z jednym bitem znaku, 8 bitami wykłanika i 23 bitami mantysy. W efekcie przy oznaczeniach jak wyżej mamy

$$|e_{obl} - e_{dok}| \leq \varepsilon |e_{obl}|$$

gdzie $\varepsilon = 2^{-24}$.

1.3 IEEE a arytmetyka systemu FriCAS

W systemie FriCAS typ `DoubleFloat` oznacza maszynowe liczby podwójnej precyzji (64-bitowe). Operacje na nich są wykonywane zgodnie z regułami IEEE. FriCAS nie daje bezpośredniego dostępu do operacji na liczbach zmiennopozycyjnych pojednczej precyzji (32-bitowych).

Typ `Float` oznacza realizowane programowo liczby wielokrotnej precyzji. Operacje na nich *nie* są zgodne z regułami IEEE.

Komentarz: Reguły IEEE mają dać możliwie duża dokładność przy reprezentacji liczb za pomocą 64-bitów. Aby osiągnąc wymaganą dokładność sprzęt musi używać dodatkowe bity (1-2) w pośrednich etapach. Przy realizacji programowej dodatkowe bity wymagałyby dodatkowego słowa (czyli na 64-bitowej maszynie dodatkowych 64-bitów) i reguły IEEE sprowadzały by się do prowadzania obliczeń z wiekszą niż wymagana dokładnością, lecz co chwilę odrzucając dodatkowo obliczoną część. Przy realizacji sprzętowej ma to sens bo dodatkowe bity niewiele kosztują zaś wynik trzeba zmieścić w dostępnej pamięci czyli 64 bitach. Przy realizaji programowej prawie zawsze prowadzenie obliczeń z większą ilością bitów a mniej dokładnym zaokrąglaniem daje wiekszą dokładność i szybkość.

Komentarz: Operacje na typie `Float` wymagają znacznie wiecej czasu (rzędu 100 razy więcej lub gorzej) niż operacjie na typie `DoubleFloat` o ile operacje na `DoubleFloat` są zaprogramowane z dbałością o efektywność.

1.4 Powtarzalność obliczeń

Jedną z motywacji arytmetki IEEE było zapewnienie że ten sam ciąg operacji zmiennopozycyjnych na różnych komputerach zgodnych z IEEE da ten sam wynik (tzn. każdy bit wyniku będzie się zgadzał). Niestety, w praktyce to może łatwo zawieść. Mianowicie, programów nie pisze się jako ciągów instrukcji w języku maszynowym (wtedy raczej by nie działały na komputerze innego typu), lecz kompilator tłumaczy program w języku wyższego rzędu na język maszynowy. Na różnych komputerach różne ciągi instrukcji mogą dawać maksymalną szybkość obliczeń. Np. operacje arytmetyki IEEE nie są łączne, kompilator korzystając z reguł łączności może zmienić kolejność obliczeń i łatwiej użyć poprzednio obliczony wynik czy wykonać równolegle więcej operacji. Nowe komputery udostępniają operację mnożenia z dodawaniem (ang. multiply and add), gdzie jedna instrukcja oblicza wyrażenie typu $ab + c$. Przy takim obliczeniu zaokrąglenie wykonuje się dopiero na końcu. Może to dać zupełnie inny wynik niż oddzielne operacje.

Warto tu też wspomnieć funkcje biblioteczne. Praktyczne obliczenia używają zwykle sporo funkcji bibliotecznych, jeśli biblioteka od innego autora używa inny ciąg instrucji (czyli że jest to faktycznie inna biblioteka a nie kopia) to zwykle też będą różnice w wynikach.

W języku Java próbowano zapewnić zgodność wyników na różnych komputerach. Piszę próbowano, bo były przeoczenia i w niektórych (bardzo rzadkich) przypadkach wyniki się nie zgadzały. Jednakże koszt zgodności jest spory, zwykle inne języki które nie dają takich zapewnień pozwalają uzyskać znacznie większą szybkość obliczeń zmiennopozycyjnych.

1.5 Podsumowanie

Artmetyka IEEE wyrosła z doświadczeń z wcześnieszymi systemami i badań teoretycznych. Reguła zaokrąglania IEEE może się wydawać skomplikowana i trudna do realizacji. Jednakże zwykle daje ona mniejsze błędy niż alternatywy. Np. w serii 360 (w pełni zgodni następcy są ciągle w produkcji) firma IBM użyła podstawy 16 i regułę obcinania niereprezentowalnych bitów wyniku. Wtedy wydawało się że pozwoli to uproszczenie sprzętu przy co najmniej tak dobrych wynikach jak alternatywy. Gdy jednak po wejściu tych maszyn do użycia przeprowadzono dokładniejszą analizę to okazało się że arytmetyka IBM daje dokładność porównywalną z artmetyką IEEE używającą 3 bity mniej. Innymi słowy 32-bitowe liczby IBM dawały dokładność porównywalną z 29-bitowym IEEE, zaś 32-bitowe liczby IEEE dają istotnie lepszą dokładność. Warto też powiedzieć że podstawa 2 ma numerycznie nieco lepsze własności niż inne podstawy (np. podstawa 10).

2 Propagacja błędów

Aby przeanalizować wpływ błędów zaokrągleń zakładamy specificzną postać programu, mianowicie tak zwany program liniowy. Zakładamy że dane są węzły obliczniowe (zmienne) v_i, $i = 1, ..., n$. Pierwsze m zmiennych to dane wejściowe x_i, tzn. $v_i = x_i$ dla $i = 1, ..., m$. Dla $i > m$ zakładamy że dane są funkcje gładkie f_i takie że w dokładnych obliczeniach

$$v_i = f_i(v_1, \ldots, v_{i-1})$$

Zwykle funkcje f_i zależą tylko od niewielu zmiennych. Np. f_i to operacja arytmetyczna na dwu poprzednich zmiennych. Zaokrąglenia modelujemy w ten sposób że piszemy

$$v_i = f_i(v_1, \ldots, v_{i-1}) + e_i$$

gdzie e_i jest błędem powstałym przy oblicznaniu f_i. A więc przybliżone obliczanie v możemy modelować jako dokładne obliczanie funkcji wektorowej n zmiennych, gdzie zmienne to x_i dla $i = 1, \ldots, m$ i e_i dla $i = m + 1, \ldots, n$. Oznaczając tą funkcję przez h mamy

$$h_i(x, e) = f_i(v_1, \ldots, v_{i-1}) + e_i.$$

gdzie pośrednie v_i spełniają zależności podane wyżej.

Niech teraz w_0 oznacza dokładnie obliczony wynik, tzn. taki wektor wartości który otrzymamy gdy wszystkie $e_i = 0$. Wtedy

$$w_0 = h(x, 0)$$

Ogólniej, przez w_t oznaczmy wektor wartości który otrzymamy gdy błąd w kroku i to te_i:

$$w_t = h(x, te).$$

Oczywiście w_1 to faktycznie obliczony wynik.

Jako że f_i są gładkie to h też jest funkcją gładką i mamy

$$|(w_1 - w_0)_i| \leq \int_0^1 |\sum_k (\partial_{e_k} h(x, te))_i e_k| dt.$$

Obliczenia są dobrze uwarunkowane jeśli istnieje mała stała δ, nie za duża stała M i stałe $g_{i,k}$ takie że

$$|e_i| \leq \delta |(w_0)_i|,$$

$$\sup_{t \in [0,1]} |(\partial_{e_k} h(x, te))_i| \leq g_{i,k}$$

i

$$\sum_i g_{i,k} |(w_0)_k| \leq M |w_i|.$$

Druga nierówność wyżej implikuje że

$$\int_0^1 |(\sum_k \partial_{e_k} h(x, te) e_k)_i| dt \leq \sum_k g_{i_k} |e_k|$$

czyli z nierówności wyżej otrzymujemy:

$$|(w_1 - w_0)_i| \leq \sum_k g_{i,k} |e_k| \leq \sum_k g_{i,k} \delta |(w_0)_k| \leq \delta M |w_i|.$$

Innymi słowy błąd względny wyniku jest ograniczony przez δM.

Oszacowanie które otrzymaliśmy jest bardzo zadowalające, niestety sprawdzenie założeń może być kłopotliwe. Warunek

$$|e_i| \leq \delta |(w_0)_i|$$

sprawia mniej problemu, jest to założenie że pojedyncza operacja nie wprowadza dużego błędu. Dla arytmetki IEEE zwykle jest ono spełnione, dokładniej jeśli pozostajemy w zakresie normalnych liczb to

$$|e_i| \leq \epsilon |v_i|$$

Jeśli nie popełnimy grubego błędu to $|v_i| < 2|(w_0)_i|$ i oszacowanie będzie spełnione z $\delta = 2\epsilon$. Jeśli spełnione są pozostałe założenia z rozsądnie dużym M to indukcyjnie można pokazać że $|v_i| < 2|(w_0)_i|$.

Założenie

$$\sup_{t\in[0,1]} |(\partial_{e_k} h(x,te))_i| \leq g_{i,k}$$

może sprawiać dwa problemy. Jeden taki że punkt gdzie obliczamy f_i się zmienia. Jeśli pochodne w punktach odpowiadających dokładnym obliczeniom mają dobre ograniczenia to zmiana punktu raczej nie prowadzi do problemu, ale trudno tu udowadnić jakiś silny i ogólny wynik. Drugi, istotniejszy problem to to że h jest złożeniem wielu funkcji. Nawet jak każda z osobna ma niezbyt dużą pochodną to złożenie może mieć bardzo duża pochodną. Warunek ograniczoności pochodnych $\partial_{e_k} h(x,te)$ zwykle nazywa się stabilnością obliczeń. Czyli jeśli obliczenia są stabilne (co jest matematyczną własnością funkcji które mamy obliczyć i wybranego sposobu obliczania) to błędy zaokrągleń nie stanowią problemu. Dalej rozpatrujemy kilka przykładów obliczeń stabilnych i niestabilnych (zaczynając od niestabilnych).

Przykład. Rozważny następujące obliczenie w arytmetyce zespolej:

$$z_{i+1} = z_i^2$$

zaczynając z_0 (dobrze myśleć że $|z_0| = 1$). Indukcyjnie pokazujemy że

$$z_i = z_0^{2^i}.$$

Czyli

$$\partial_{z_0} z_n = 2^n \frac{z_n}{z_0}.$$

Innymi słowy dla dużych n możemy oczekiwać wykładniczego narastania błędów. W powyższym przykładzie każda metoda obliczania z_n będzie prowadziła do znacznego narastania błędu i jedynym rozwiązaniem jest użycie dostatecznie dużej precyzji obliczeń.

Przykład. Niech

$$w_i = (x - 1 + \frac{1}{i})(x + 1 - \frac{1}{i}),$$

$$P(x) = \prod_{i=2}^{n} \frac{w_i(x)}{w_i(0)},$$

$$A = \frac{1}{4} \begin{pmatrix} 11 & 20 \\ 6 & -11 \end{pmatrix}.$$

Chcemy obliczyć $P(A)$. Mamy

$$w_i' = (x^2 - (1 - \frac{1}{i})^2)' = 2x$$

czyli $|w_i'|$ nie przekracza 1 dla $x \in [-\frac{3}{8}, \frac{3}{8}]$. Widać też że dla $x \in [-\frac{3}{8}, \frac{3}{8}]$ mamy

$$|w_i(x)| = |(1 - \frac{1}{i})^2 - x^2| \geq (1 - \frac{1}{i})^2 - x^2 \geq \frac{1}{4} - \frac{9}{64} = \frac{5}{64}.$$

Ze wzoru na pochodną logarytmiczną otrzymujemy

$$P(x)' = P(x)\sum_{i=2}^{n}\frac{w_i'(x)}{w_i(x)}$$

czyli

$$|P(x)'| \leq (n-2)\frac{64}{5}|P(x)|$$

dla $x \in [-\frac{3}{8}, \frac{3}{8}]$. Wartości $P(x)$ mogą być dość duże, ale dla n rzędu 200 będziemy w zakresie liczb zmiennopozycyjnych. Dokładniej, jako że $P(0) = 1$ z oszacowania pochodnej wynika że $|P(x)| \leq \exp(\frac{2(n-2)}{3})$ dla $x \in [-\frac{1}{4}, \frac{1}{4}]$, co dla $n = 200$ daje w przybliżeniu $2.1 \cdot 10^{57}$.

Diagonalizując A widzimy że

$$A = \frac{1}{4}\begin{pmatrix} 3 & 5 \\ 1 & 2 \end{pmatrix}^{-1}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 3 & 5 \\ 1 & 2 \end{pmatrix}.$$

Czyli

$$P(A) = \begin{pmatrix} 3 & 5 \\ 1 & 2 \end{pmatrix}^{-1}\begin{pmatrix} P(\frac{1}{4}) & 0 \\ 0 & P(\frac{-1}{4}) \end{pmatrix}\begin{pmatrix} 3 & 5 \\ 1 & 2 \end{pmatrix}.$$

Oznacza to że pochodne $P(A)$ po elementach A są rzędu wielkości $P'(\frac{1}{4})$ co jest rzędu $nP(A)$. A więc oczekujemy że obliczanie $P(A)$ nie powinno sprawiać problemu dla n rzędu 200.

Jednakże, popatrzmy na

$$Q(x) = \prod_{i=2}^{n}(x + 1 - \frac{1}{i}).$$

Mamy $P(x) = c_n Q(x)Q(-A)$ dla odpowiedniej stałej c_n i można by próbować obliczać $P(A)$ jako $c_n Q(A)Q(-A)$. Jak się okaże nie jest to dobry pomysł. Mianowicie, skoro A ma wartości własne $\frac{1}{4}$ i $\frac{-1}{4}$ to pochodna $Q'(A)$ jest rzedu $\max(Q'(\frac{1}{4}), Q'(\frac{-1}{4}))$.

Mamy $(x + 1 - \frac{1}{i})' = 1$ i dla $x \in [-\frac{3}{8}, \frac{3}{8}]$ mamy

$$0 \leq (x + 1 - \frac{1}{i}) \leq \frac{3}{8} + 1 - \frac{1}{i} \leq \frac{11}{8}.$$

A więc ze wzoru na pochodną logarytmiczną otrzymujemy

$$Q'(x) = Q(x)\sum_{i=2}^{n}\frac{(x + 1 - \frac{1}{i})'}{x + 1 - \frac{1}{i}} \geq Q(x)\sum_{i=2}^{n}\frac{1}{\frac{11}{8}} = \frac{8(n-2)}{11}Q(x).$$

Analogicznie jak dla P otrzymujemy też podobne oszacowanie z góry. Dla nas ważne jest że oszacowania z góry nie da się istotnie poprawić. Teraz popatrzmy jak duże jest $Q(x)$. Dokładniej, interesuje nas

$$\frac{Q(x)}{Q(-x)} = \prod_{i=2}^{n}\frac{x + 1 - \frac{1}{i}}{-x + 1 - \frac{1}{i}}$$

Łatwo zauważyć że przy ustalonym x czynniki produktu maleją wraz z i. A więc

$$\frac{Q(\frac{1}{4})}{Q(\frac{-1}{4})} \geq \left(\frac{1+\frac{1}{4}}{1-\frac{1}{4}}\right)^{n-2}.$$

Dla $n = 200$ daje to w przybliżeniu $8.43 \cdot 10^{43}$. Dalej, jeśli błąd przy obliczeniu $Q(-A)$ jest rzędu $\delta\|Q(-A)\|$ to oczekujemy że błąd $Q(A)Q(-A)$ bedzie rzędu

$$\delta\|Q'(A)\|\|Q(-A)\| \approx nQ(1/4)^2$$

co po podzielniu przez normę $\|P(A)\|$ daje błąd względny rzędu

$$\delta\frac{c_n\|Q'(A)\|\|Q(-A)\|}{\|P(A)\|} \approx n\delta\frac{c_nQ(\frac{1}{4})^2}{|P(\frac{1}{4})|} = n\delta\frac{Q(\frac{1}{4})}{Q(\frac{-1}{4})}.$$

A więc skoro ostatni ułamek ma bardzo dużą wartość to oczekujemy bardzo dużego błędu.

Przykład. Popatrzy na błąd obliczania wielomianu Wilkinsona

$$p(x) = (x-1)(x-2)\dots(x-20) = \sum_{i=0}^{n} a_i x^i$$

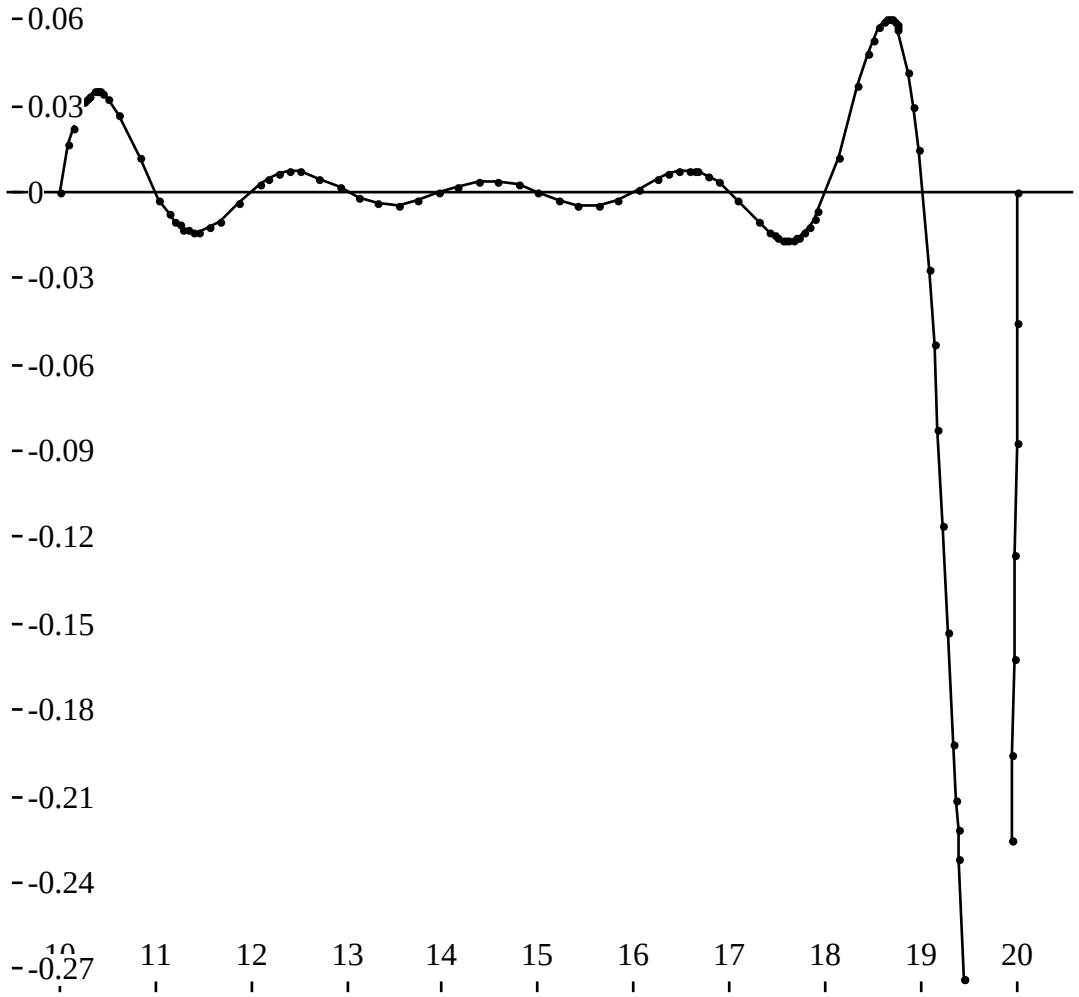
Przy użyciu wersji z sumą można oczekiwać że każdy wyraz sumy da błąd proporcjonalny do jego wielkości czyli zgrubnie

$$\sum |a_i||x_i| = (x+1)(x+2)\dots(x+20) = p(-x)$$

dla nieujemnych x. Równość wyżej wynika stąd że wyrażenie w środku po rozwinięciu da nieujemne współczynniki. Lecz ostatnia równość pokazuje że te współczynniki mają tą samą wartość bezwzględną co współczynniki p. A więc zgrubne oszacowanie błędu względnego to

$$u = \prod_{i=1}^{20} \frac{x+i}{x-i}.$$

W pobliżu zer p to u dąży do niekończoności dlatego patrzymy na wykres odwrotności u.



Wykres produkowany przez program przykładowy jest przeskalowany (jest to $10^{12}/u$). Jak widać z wykresu nawet daleko od zer p odwrotność jest mała czyli u może być duże (rzędu 10^{12}).

Napisałem zgrubne oszacowanie bo obliczając wyraz po wyrazie musimy też obliczyć potęgi co da dodatkowy błąd. Sumując liczymy na to że przeciętnie błąd sumy jest mniejszy niż suma błędów. Stosując do obliczenia schemat Hornera powinniśmy dostać nieco mniejszy błąd. Ale też jest jasne że trudność wyznaczania zer p jest związana z tym że trudno jest obliczać wartości p.

Przykład. Chcemy obliczyć

$$h = \prod_{i=1}^{n} \lambda_i$$

gdzie λ_i są liczbami. Ze wzoru na pochodną logarytmiczną wynika że

$$\partial_{\lambda_i} h = \frac{1}{\lambda_i} h$$

czyli pochodne w naszym oszacowaniu błędu będą dobrze oszacowane o ile dobrze oszacujemy h. Lecz z własności arytmetyki IEEE mamy

$$a * b \leq (1+\delta)(ab)$$

gdzie $*$ oznacza maszynowe mnożenie, zaś $\delta = \epsilon$ gdzie ϵ jest stałą podającą dokładność arytmetyki. Wtedy h_{fl}

$$h_{fl} \leq \prod_{i=1}^{n}(1+\delta) \prod_{i=1}^{n} \lambda_i = (1+\delta)^n h \leq \exp(n\delta) h$$

gdzie h oznacza wartość dokładną zaś h_{fl} przybliżenie maszynowe. A więc o ile $n\delta$ jest małe to nasze oszacowanie się stosuje.

Przykład. Obliczamy

$$\prod_{i=1}^{n} U_i v$$

gdzie U_i są macierzami ortogonalnymi zaś v jest wektorem. Jeśli pojedynczy produkt ma oszacowanie

$$\|U_i * v - U_i v\| \leq \delta \|v\|$$

gdzie jak wyżej $*$ oszacza obliczenie maszynowe to podobnie jak wyżej dostaniemy oszacowanie

$$\|U_1 * \cdots * U_n * v - U_1 \ldots U_n v\| \leq \exp(n\delta)\|v\|.$$

Tym razem δ będzie większe (zależnie od postaci i rozmiaru macierzy), lecz w kilku ważnych przypadkach (np. dla szybkiej dyskretnej transformacji Fouriera FFT) jest tylko kilka razy większe od ϵ (czyli maszynowej dokładności).

3 Utrata dokładności

Typową sytuacją gdzie następuje utrata dokładności jest odejmowanie zbliżonych do siebie liczb.

Przykład. Na ćwiczeniach patrzyliśmy na błąd przy różniczkowaniu numerycznym. Pierwsze przybliżenie $f'(x)$ to

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

Stosując wzór Taylora mamy

$$|f(x+h) - f(x) + hf'(x)| \leq \frac{h^2}{2} \sup_{t\in[0,1]} |f''(x+th)|$$

czyli

$$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| \leq \frac{|h|}{2} \sup_{t\in[0,1]} |f''(x+th)|.$$

Oznacza to że w dokładnej arytmetyce im mniejsze $|h|$ tym lepsze przybliżenie pochodnej. Innymi słowy błąd metody maleje z $|h|$. Jednakże jeśli $f(x+h)$ i $f(x)$ są obliczone z dokładnością względną δ to błąd zaokrąglenia we wzorze wyżej jest rzędu

$$2\frac{\delta|f(x)|}{|h|}.$$

Oznacza to że bład zaokrąglenia będzie dominował nad błędem metody o ile

$$2\frac{\delta|f(x)|}{|h|} \geq \frac{|h|}{2}|f''(x)|$$

Powyżej zastąpiliśmy $\sup_{t\in[0,1]} |f''(x+th)|$ co nie powinno istotnie zmienić wyniku dla regularnych f i małych $|h|$. Warunek wyżej można zapisać jako

$$|h|^2 \leq \delta\frac{4|f(x)|}{|f''(x)|}.$$

A więc zakładając że f jest naturalnie przeskolowane, czyli ułamek po prawej stronie jest rzędu 1, to $|h|$ rzędu pierwiastka z δ da nam optymalną dokładność. Zakładając że δ jest rzędy elpilona maszynowego, czyli 2^{-53} otrzymujemy $|h|$ rzędu 2^{-26} czyli w przybliżeniu $1.4\cdot10^{-8}$. Przy tym błąd jest tego samego rzędu.

Przybliżając $f'(x)$ za pomocą

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

z wzór Talora trzeciego rzędu mówi że

$$|f(x+h) - hf'(x) - h^2f''(x)| \leq \frac{|h|^3}{6} \sup_{t\in[0,1]} |f'''(x+th)|$$

Odejmując podobny wzór dla $-h$ mamy

$$|(f(x+h) - f(x-h)) - 2hf'(x)| \leq \frac{|h|^3}{3} \sup_{t\in[-1,1]} |f'''(x+th)|$$

co daje

$$|\frac{f(x+h)-f(x-h)}{2h} - f'(x)| \leq \frac{|h|^2}{6} \sup_{t\in[-1,1]} |f'''(x+th)|.$$

Błąd zaokrąglenia jest rzędu $\frac{\delta|f(x)|}{|h|}$ (tym razem dodatkowo dzielimy przez 2). Jak poprzednio szacując wartości f i pochodnych przez wartość w x wychodzi że błąd zaokrąglenia będzie dominował dla

$$|h|^3 \leq \delta \frac{6|f(x)|}{|f'''(x)|}.$$

Znowu zakładając że f jest naturalnie przeskalowana i ułamek po prawej stronie ma wartość rzędu 1 dostajemy że optymalne h jest rzędu $\delta^{\frac{1}{3}}$ czyli rzędu $4.8 \cdot 10^{-6}$. A więc tym razem lepiej użyć większe h. Tym razem błąd względny jest rzędu $|h|^2$, czyli $2.3 \cdot 10^{-11}$, a więc istotnie mniejszy niż poprzednio.

Komentarz: Pokazuje to że nawet jak nie jest potrzebna bardzo duża dokładność końcowych wyników różniczkowanie numeryczne wymaga dużej dokładności f, a więc lepiej użyć liczby podwójnej precyzji. Wiele tekstów proponuje zastąpienie pochodnych różnicami skończonymi. Z powyższego widać że może to doprowadzić do utraty dokładności. Jeśli różnice skończone przy użyciu liczb 64-bitowych dają wystarczającą dokładność to jest prawdopodobne że pochodne i liczby 32-bitowe też dałyby zadowalającą dokładność przy niższym czasie pracy komputera.

Komentarz: Przy utracie dokładności jest paradoks: samo odejmowanie prowadzące do utraty dokładności zwykle jest wykonywane dokładnie. Mianowicie, przy utracie dokładności skracają się (są zerami) najbardziej znaczące bity wyniku, co oznacza że wynik można dokłanie reprezentować w mniejszej ilości bitów niż argumenty. Jedyna możliwość błędu przy odejmowaniu z utratą dokładności jest wtedy gdy wartość bezwzględna wyniku jest zbyt mała by ją reprezentować jako liczbę normalną i komputer zastąpi wynik przez zero. Natomiat błąd pochodzi z poprzednich kroków i zwykle propaguje się do wyniku odejmowania. Czyli błąd bezwględny jest podobnego rzędu jak przed odejmowaniem. Ale gdy wynik odejmowania ma małą wartość bezwzględną to błąd względny rośnie: czyli tu jest utrata dokładności. Napisałem wyżej zwykle bo jest co najmniej jeden trikowy algorytm który produkuje dane tak by przy odejmowaniu błąd z poprzednich kroków się dokładnie odjął.