Obliczenia Naukowe - Lista nr 1

Makysmilian Piotrowski

1 Zadanie 1

1.1 Epsiliony maszynowe

Opis problemu: Problem polega na wyznaczeniu liczby macheps dla trzech typów zmiennopozycyjnych języka Julia: Float16, Float32, Float64.

Rozwiązanie: Obliczeń dokonałem iteracyjnie. Rozpoczynałem od x = 1 w konkretnym typie. Następnie dzieliłem x w pętli przez dwa. Najmniejszy znaleziony x spełniający $1.0 + x > 1.0$ zapisałem jako macheps. Skrypt znajduje się w pliku l1_z1a.jl.

Wyniki i interpretacja:

Typ	macheps	eps(Typ)
Float16	0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	2.220446049250313e-16	2.220446049250313e-16

Tabela 1: Wygenerowane wartości macheps i wartości funkcji eps w zależności od podanego typu.

Wygenerowane wartości macheps były identyczne z wartościami uzyskanymi funkcją eps. Wraz ze zwiększeniem precyzji macheps maleje.

Wnioski: Jeśli dodamy liczbę mniejszą niż epsilon maszynowy do 1.0 przy operowaniu na liczbach zmiennoprzecinkowych, otrzymamy w wyniku 1.0. Epsilion maszynowy jest równy dwukrotności precyzji arytmetyki oznaczanej na wykładzie przez ϵ.

1.2 Liczba maszynowa eta

Opis problemu: Problem polega na wyznaczeniu liczby maszynowej *eta*, taką że $eta > 0.0$, dla trzech typów zmiennopozycyjnych języka Julia: Float16, Float32, Float64.

Rozwiązanie: Obliczyłem iteracyjnie liczbę maszynową eta. Najmniejszy znaleziony x konkretnego typu spełniający $x > 0.0$ zapisałem jako eta. Skrypt znajduje się w pliku l1_z1b.jl.

Wyniki i interpretacja:

Typ	eta	nextfloat(Typ(0.0))	floatmin(Typ)
Float16	6.0e-8	6.0e-8	6.104e-5
Float32	1.0e-45	1.0e-45	1.1754944f-38
Float64	5.0e-324	5.0e-324	2.2250738585072014e-308

Tabela 2: Wygenerowane liczby maszynowe eta i wartości funkcji nextfloat(0.0), floatmin w zależności od podanego typu.

Wygenerowane wartości eta były identyczne z wartościami uzyskanymi funkcją nextloat(0.0). Wraz ze zwiększeniem precyzji macheps maleje.

Wnioski: W arytmetyce zmiennopozycyjnej IEEE 754 liczby odpowiednio bliskie 0 są traktowane jako 0. Możliwe jest jednak zareprezentowanie liczby dużo mniejszej niż macheps z Tabeli 1.
Wartości eta i nextfloat(0) Float32, Float64 są tego samego rzędu wielkości co MINsub z wykładu dla arytemtyki single, double. Są one zdenormalizowane. Wartości floatmin Float32, Float64 są tego samego rządu co wielkości MINnor z wykładu dla arytmetyki single, double. Są znormalizowane.

1.3 MAX

Opis problemu: Problem polega na obliczeniu liczby (MAX), dla trzech typów zmiennopozycyjnych języka Julia: Float16, Float32, Float64.

Rozwiązanie: Obliczyłem liczbę (MAX). Zrobiłem to powiększając liczbę x konkretnego typu do momentu kiedy isinf(x) zaczęło zwracać true przy najmniejszym możliwym powiększeniu x. Taką liczbę x zapisałem jako MAX. Zapisałem też wartości wywołań funkcji floatmax, oraz wartości ze wspomnianego na liście zadań raportu. Skrypt znajduje się w pliku l1_z1c.jl.

Wyniki i interpretacja:

Typ	MAX	floatmax(Typ)	raport
Float16	6.55e4	6.55e4	
Float32	3.4028235e38	3.4028235e38	3.4e38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.8 E308

Tabela 3: Wygenerowane MAX, wartości funkcji floatmax oraz dane z raportu w zależności od podanego typu.

Wygenerowane wartości MAX były tych samych rzędów wielkości, co te z pozostałych źródeł.

Wnioski: Maksymalne wartości liczb zmiennoprzecinkowych IEEE 754 znacznie różnią się między typami.

2 Zadanie 2

Opis problemu: Problem polega na sprawdzeniu tezy Kahana, że epsilon maszynowy (macheps) można otrzymać obliczając wyrażenie $3(4/3-1)-1$ w arytmetyce zmiennopozycyjnej.

Rozwiązanie: Sprawdziłem tezę eksperymentalnie dla Float16, Float32, Float64.

Wyniki i interpretacja:

Typ	wyrażenie	eps(Typ)
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Tabela 4: Wartości wyrażenia $3*(4/3-1)-1$ oraz wartości funkcji eps(typ) w zależności od typu.

Otrzymane wartości mają bezwzględną wartość identyczną z epsilionem maszynowym. Zatem stwierdzenie Kahana jest prawdziwe. Epsilion maszynowy można uzyskać obliczając $|3*(4/3-1)-1|$.

Wnioski: Wyrażenia równe 0 w zwykłej arytmetyce mogą w arytmetyce zmiennopozycyjnej zwrócić wartość różną od zera.

3 Zadanie 3

Opis problemu: Problem polega na sprawdzeniu, że w arytmetyce Float64 liczby zmiennopozycyjne są równomiernie rozmieszczone w [1, 2] z krokiem $\delta = 2^{-52}$, oraz wyznaczeniu tego kroku dla prze-

działów $[\frac{1}{2}, 1]$ oraz [2,4].

Rozwiązanie: Używając funkcji nextfloat, prevfloat i bitstring sprawdziłem czy rzeczywiście, w języku Julia kolejne liczby z przedziału [1,2] różnią się o 2^{-52}. Następnie porównywałem kolejne liczby z przedziłów $[\frac{1}{2}, 1]$, [2,4], aby wyznaczyć ich krok.

Wyniki i interpretacja: Kolejne liczby z przedziału [1,2] rzeczywiście różnią się o 2^{-52}. Liczby z przedziału $[\frac{1}{2}, 1]$ różnią się o 2^{-53}, bo używają tej samej liczby bitów aby opisać dwa razy mniejszy przedział niż [1,2].
Liczby z przedziału [2,4] różnią się o 2^{-51}, bo używają tej samej liczby bitów aby opisać dwa razy większy przedział niż [1,2].

Wnioski: Liczby pomiędzy 0.5 i 1 można przedstawić jako $x = 0.5 + k\delta$, gdzie $k = 1, 2, ..., 2^{52} - 1$ i $\delta = 2^{-53}$. Liczby pomiędzy 2 i 4 można przedstawić jako $x = 2 + k\delta$, gdzie $k = 1, 2, ..., 2^{52} - 1$ i $\delta = 2^{-51}$.
Przedziały bliższe zera mają mniejszy krok, niż te dalej od zera w IEEE 754.

4 Zadanie 4

Opis problemu: Problem polega na znalezieniu (najmniejszej) liczby Float64 w przedziale $1 < x < 2$, takiej, że $x * (1/x) \neq 1$.

Rozwiązanie: Znalazłem liczbę x w przedziale $1 < x < 2$ spełniającą $fl(xfl(1/x)) \neq 1$ iterując się po kolejnych x, zaczynając od x = 1, przy pomocy funkcji nextfloat. Pierwsza znaleziona w ten sposób liczba jest zarazem najmniejsza. Skrypt znajduje się w pliku l1_z4.jl.

Wyniki i interpretacja:

x = 1.000000057228997
bitstring(x) = 0011111111110000000000000000000000000011110101110010111111100101010

Wnioski: Skończona precyzja sprawia, że prawdziwe równości takie jak $x * (1/x) = 1$ mogą być fałszywe w arytemtyce zmiennopozycyjnej.

5 Zadanie 5

Opis problemu: Problem polega na napisaniu skryptu w Julii obliczającego sumę iloczynu wektorowego x, y na cztery zdefiniowane na Liście sposoby i porównaniu wyników. Należy wykonać dla Float32 i Float64.
x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]
y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049].
Prawidłowa wartość sumy (dokładność do 15 cyfr): $-1.00657107000000) * 10^{-11}$

Rozwiązanie: Napisałem skrypt obliczający sumę iloczynu skalarnego wektorów x,y na sposoby opisane na Liście. Skrypt znajduje się w pliku l1_z5.jl.

Wyniki i interpretacja:

Algorytm	Float32	Float64
W przód(a)	-0.4999443	1.0251881368296672e-10
W tył(b)	-0.4543457	-1.5643308870494366e-10
max→min(c)	-0.5	0.0
min→max(d)	-0.5	0.0

Tabela 5: Wyniki sumowania w zależności od algorytmu i używanego typu.

Wyniki sumowania iloczynu sklalarnego różnią się znacząco w zależności od kolejności wykonywania działań i używanego typu zmiennoprzecinkowego.

Wnioski: Z pozoru równoważne algorytmy mogą dawać różne wynik dla tych samych danych, w zależności od kolejności wykonywania działań i używanych typów danych.

6 Zadanie 6

Opis problemu: Problem polega na obliczeniu wartości funkcji

$$f(x) = \sqrt{x^2 + 1} - 1$$

,

$$g(x) = x^2/(\sqrt{x^2 + 1} + 1)$$

dla kolejnych wartości argumentu x = 8^{-1}, 8^{-2}, ... oraz na ustaleniu która funkcja daje wiarygodniejsze wyniki.

Rozwiązanie: Obliczyłem wartości f(x), g(x) dla wybranych x, aby zilustrować jak zmieniają się w zależności od wartości argumentu. Skrypt znajduje się w pliku l1_z6.jl.

Wyniki i interpretacja:

x	f(x)	g(x)
8^{-1}	0.0077822185373186414	0.0077822185373187065
8^{-2}	0.00012206286282867573	0.00012206286282875901
8^{-3}	1.9073468138230965e-6	1.907346813826566e-6
8^{-4}	2.9802321943606103e-8	2.9802321943606116e-8
8^{-5}	4.656612873077393e-10	4.6566128719931904e-10
8^{-6}	7.275957614183426e-12	7.275957614156956e-12
8^{-7}	1.1368683772161603e-13	1.1368683772160957e-13
8^{-8}	1.7763568394002505e-15	1.7763568394002489e-15
8^{-9}	0.0	2.7755575615628914e-17
8^{-10}	0.0	4.336808689942018e-19
8^{-20}	0.0	3.76158192263132e-37
8^{-30}	0.0	3.2626522339992623e-55
8^{-40}	0.0	2.8298997121333476e-73
8^{-50}	0.0	2.4545467326488633e-91
8^{-60}	0.0	2.1289799200040754e-109
8^{-70}	0.0	1.8465957235571472e-127
8^{-80}	0.0	1.6016664761464807e-145
8^{-90}	0.0	1.3892242184281734e-163
8^{-100}	0.0	1.204959932551442e-181
8^{-120}	0.0	9.065110999561118e-218
8^{-140}	0.0	6.819831532519088e-254
8^{-160}	0.0	5.1306710016229703e-290
8^{-180}	0.0	0.0
8^{-200}	0.0	0.0

Tabela 6: Wartości f(x), g(x) w zależności od x

Dla $x \geqslant 8^{-8}$ funkcje zgodnie z oczekiwaniami zwracają coraz mniejsze wartości wraz z obniżaniem wartości x. Funkcja f jednak już dla $x = 8^{-9}$ podaje błędny wynik f(x) = 0.
Dzieje się tak, ze względu na utratę dokładności przy dodawaniu bardzo małej liczby x do relatywnie dużej liczby 1.
Algorytm g(x) operuje na x^2, którego liczby po przecinku nie są w podobny sposób tracone poprzez dodanie do 1.

Biorąc to wszystko pod uwagę, wyniki g(x) są bardziej wiarygodne.

Wnioski: Wyniki funkcji w IEEE 754 mogą być mniej lub bardziej precyzyjne ze względu na sposób ich zapisu. Należy uważać na dodawanie dużych liczb do bardzo małych liczb, bo może to doprowadzić do utraty precyzji.

7 Zadanie 7

Opis problemu: Problem polega na policzeniu przybliżonej wartości pochodnej funkcji $f(x) = sinx + cos3x$ przy pomocy wzoru $\tilde{f}'(x) = \frac{f(x_0+h)-f(x_0)}{h}$ dla $x_0 = 1$, $h = 2^{-n}$ (n = 1,2,...,54) w języku Julia w arytmetyce Float64. Należy przeanalizować wyniki.

Rozwiązanie: Policzyłem przybliżenia pochodnych zgodnie ze specyfikacją zadania. Policzyłem też $f'(1)$ wzorem $f'(x) = cos(x) - 3 * sin(3x)$, i porównałem z wartościami $\tilde{f}'(1)$. Skrypt znajduje się w l1_z7.jl.

Wyniki i interpretacja:

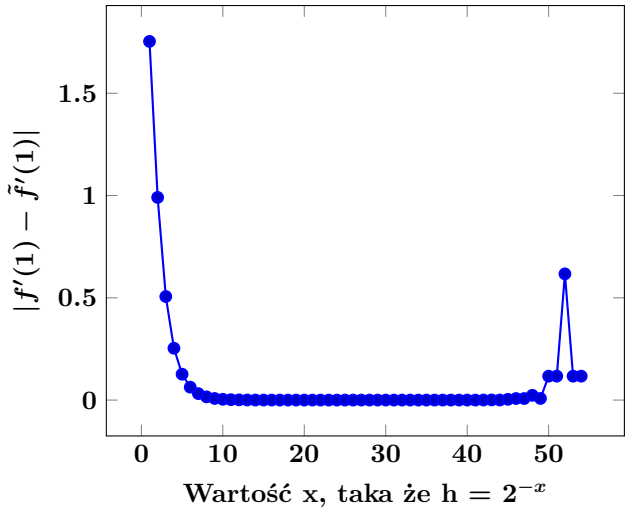
Tabela 7: Przybliżona pochodna w punkcie $x_0 = 1$ oraz jej odległość od faktycznej pochodnej w punkcie $x_0 = 1$ w zależności od h.

h	$\tilde{f}'(1)$	$\lvert f'(1) - \tilde{f}'(1)\rvert$
2^{-1}	1.8704413979316472	1.753499116243109
2^{-2}	1.1077870952342974	0.9908448135457593
2^{-3}	0.6232412792975817	0.5062989976090435
2^{-4}	0.3704000662035192	0.253457784514981
2^{-5}	0.24344307439754687	0.1265007927090087
2^{-6}	0.18009756330732785	0.0631552816187897
2^{-7}	0.1484913953710958	0.03154911368255764
2^{-8}	0.1327091142805159	0.015766832591977753
2^{-9}	0.1248236929407085	0.007881411252170345
2^{-10}	0.12088247681106168	0.0039401951225235265
2^{-11}	0.11891225046883847	0.001969968780300313
2^{-12}	0.11792723373901026	0.0009849520504721099
2^{-13}	0.11743474961076572	0.0004924679222275685
2^{-14}	0.11718851362093119	0.0002462319323930373
2^{-15}	0.11706539714577957	0.00012311545724141837
2^{-16}	0.11700383928837255	6.155759983439424e-5
2^{-17}	0.11697306045971345	3.077877117529937e-5
2^{-18}	0.11695767106721178	1.5389378673624776e-5
2^{-19}	0.11694997636368498	7.694675146829866e-6
2^{-20}	0.11694612901192158	3.8473233834324105e-6
2^{-21}	0.1169442052487284	1.9235601902423127e-6
2^{-22}	0.11694324295967817	9.612711400208696e-7
2^{-23}	0.11694276239722967	4.807086915192826e-7
2^{-24}	0.11694252118468285	2.394961446938737e-7
2^{-25}	0.116942398250103	1.1656156484463054e-7
2^{-26}	0.11694233864545822	5.6956920069239914e-8
2^{-27}	0.11694231629371643	3.460517827846843e-8
2^{-28}	0.11694228649139404	4.802855890773117e-9
2^{-29}	0.11694222688674927	5.480178888461751e-8
2^{-30}	0.11694216728210449	1.1440643366000813e-7

Kontynuacja na kolejnej stronie

Tabela 7 – Kontynuacja kolejnej strony

h	$\tilde{f}'(1)$	$\lvert f'(1) - \tilde{f}'(1)\rvert$
2^{-31}	0.11694216728210449	1.1440643366000813e-7
2^{-32}	0.11694192886352539	3.5282501276157063e-7
2^{-33}	0.11694145202636719	8.296621709646956e-7
2^{-34}	0.11694145202636719	8.296621709646956e-7
2^{-35}	0.11693954467773438	2.7370108037771956e-6
2^{-36}	0.116943359375	1.0776864618478044e-6
2^{-37}	0.1169281005859375	1.4181102600652196e-5
2^{-38}	0.116943359375	1.0776864618478044e-6
2^{-39}	0.11688232421875	5.9957469788152196e-5
2^{-40}	0.1168212890625	0.0001209926260381522
2^{-41}	0.116943359375	1.0776864618478044e-6
2^{-42}	0.11669921875	0.0002430629385381522
2^{-43}	0.1162109375	0.0007313441885381522
2^{-44}	0.1171875	0.0002452183114618478
2^{-45}	0.11328125	0.003661031688538152
2^{-46}	0.109375	0.007567281688538152
2^{-47}	0.109375	0.007567281688538152
2^{-48}	0.09375	0.023192281688538152
2^{-49}	0.125	0.008057718311461848
2^{-50}	0.0	0.11694228168853815
2^{-51}	0.0	0.11694228168853815
2^{-52}	-0.5	0.6169422816885382
2^{-53}	0.0	0.11694228168853815
2^{-54}	0.0	0.11694228168853815



Błąd $|f'(1) - \tilde{f}'(1)|$ przyjmuje wysokie wartości dla dużych h, takich jak 2^{-1}, 2^{-2}. Jest to zgodne z oczekiwaniami, bo h we wzorze na pochodną dąży do 0.
Dla początkowych h = 2^{-1}, 2^{-2},..,2^{-28} wartości błędów maleją przy zmniejszaniu h. Najmniejszy błąd otrzymałem dla $h = 2^{-28}$.
Dla mniejszych h = 2^{-29}, 2^{-30}, ... ,2^{-52} błąd pozostaje większy niż dla 2^{-28}. Ciąg wartości błędów w zależności od h jest od tego momentu niemonotoniczny.
Dzieje się tak przez utratę dokładności przy wykonywaniu operacji na małych liczbach h we wzorze na $\tilde{f}'(1)$.
Dla h = 2^{-50} przybliżona pochodna w $x_0 = 1$ przyjmuje wartość 0.
Nagły wzrost błędu widoczny na wykresie nastąpił dla h = 2^{-52} - epsilionu maszynowego typu Float64.

Wnioski: Należy wystrzegać się operowania na wartościach bliskich 0, nawet jeśli formuła matematyczna zawiera wartość zbiegającą do 0,