Obliczenia Naukowe

Sprawozdanie z laboratorium nr 1

Tomasz Niedziałek

19 października 2025

Pliki źródłowe dla zadań 1,2,4,5,6,7 znajdują się w załączonym pliku .zip

1 Zadanie 1: Rozpoznanie arytmetyki

1.1 Opis problemu

Celem zadania jest eksperymentalne wyznaczenie w języku Julia epsilonu maszynowego, najmniejszej liczby dodatniej (*eta*) oraz największej skończonej liczby (MAX) dla typów 'Float16', 'Float32' i 'Float64'. Wyniki należy porównać z wartościami zwracanymi przez wbudowane funkcje oraz danymi z plików nagłówkowych języka C.

1.2 Rozwiązanie

Napisałem w sumie 4 funkcje. Pierwsza iteracyjnie wyznacza ϵ wpierw ustawiając go na 1 w podanym do niej typie, po czym dzieli tę jedynkę przez 2, aż zostanie spełnione równanie $1+\epsilon = 1$, wtedy zwraca wartość ϵ z poprzedniej iteracji. Druga podobnie dzieli $\eta = 1$ w danej arytmetyce, ale do momentu kiedy $\eta = 0$, po czym podobnie jak wyżej zwraca poprzednią wartość. Kolejne dwie liczą MAX. Jedna jest moim nie udanym pierwszym pomysłem - najpierw wyszukuje wartość po której kolejne przemnożenie przez 2 zwróci *inf* po czym dodaje ϵ. Było to oczywiście niepotrzebnie długie rozwiązanie. Kolejna działa poprawnie, podobnie jak wyżej, najpierw znajduje najwyższą wartość po której przemnożeniu przez 2 zwrócony zostanie *inf*, nastepnie dodaje do otrzymanej MAX, $increment = MAX/2$ jeśli $MAX + increment$ zwróciłoby *inf* $\Rightarrow increment/2$, powtarzane aż $incerment = 0$ 'wypełniając' mantysę.

1.3 Wyniki i ich interpretacja

W tabelach poniżej zestawiono uzyskane wyniki z wartościami referencyjnymi.

Tabela 1: Porównanie wyznaczonych ϵ.

Arytmetyka	Wyznaczona iteracyjnie	Z funkcji wbudowanej	Z float.h
half	0.000977	0.000977	
single	1.1920929e-7	1.1920929e-7	1.192092896e-07
double	2.220446049250313e-16	2.220446049250313e-16	2.2204460492503131e-016

Tabela 2: Porównanie wyznaczonych η.

Arytmetyka	Wyznaczona iteracyjnie	Z funkcji wbudowanej
half	6.0e-8	6.0e-8
single	1.0e-45	1.0e-45
double	5.0e-324	5.0e-324

Tabela 3: Porównanie wyznaczonych MAX.

Arytmetyka	Wyznaczona iteracyjnie	Z funkcji wbudowanej	Z float.h
half	6.55e4	6.55e4	
single	3.4028235e38	3.4028235e38	3.402823466e+38
double	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623158e+308

Odpowiedzi na pytania:

- **Związek macheps z precyzją:** macheps = 2 * ϵ, macheps to odległość między kolejnymi liczbami maszynowymi, podczas gdy ϵ to maksymalny błąd zaokrąglenia "po środku tej odległości"

- **Związek η z $MIN_{\mathbf{sub}}$:**
  $\eta = MIN_{\text{sub}}$ ponieważ osiągnęliśmy najmniejszą liczbę **nieznormalizowaną** > 0

- **Funkcje floatmin i związek z $MIN_{\mathbf{nor}}$:**
  $MIN_{\text{nor}(32)} = 1.0 * 2^{c_{min}} = 2^{-126} \approx 1.17549435 * 10^{-38}$
  $MIN_{\text{nor}(64)} = 1.0 * 2^{c_{min}} = 2^{-1022} \approx 2.2250738585072014 * 10^{-308}$
  floatmin(Float32) = 1.1754944e-38
  floatmin(Float64) = 2.2250738585072014e-308

2 Zadanie 2: Stwierdzenie Kahana

2.1 Opis problemu

Zadanie polega na eksperymentalnym sprawdzeniu w języku Julia, czy epsilon maszynowy można otrzymać, obliczając wyrażenie $3(4/3 - 1) - 1$ dla typów 'Float16', 'Float32' i 'Float64'.

2.2 Wyniki

Wyniki obliczeń dla poszczególnych typów:

Tabela 4: Porównanie $3(4/3 - 1) - 1$ i wbudowanej w Julii.

Arytmetyka	**Kahan**	**Z funkcji wbudowanej**
half	-0.000977	0.000977
single	1.1920929e-7	1.1920929e-7
double	-2.220446049250313e-16	2.220446049250313e-16

2.3 Wnioski

Wyniki Kahana są poprawne, jednak dla *half* i *double* dostajemy wartości ujemne. Wynika to z zaokrąglania *round to even* $4/3$, które dla tych arytmetyk daje wartość nieco mniejszą niż realne $4/3$. Przez co $3 * 1/3 - 1$ ląduje po ujemnej stronie od 0. Przeciwna zależność zachodzi dla *single* gdzie wartość ta jest nieco większa od rzeczywistej.

3 Zadanie 3: Rozmieszczenie liczb zmiennopozycyjnych

3.1 Opis problemu

Celem jest weryfikacja równomiernego rozmieszczenia liczb zmiennopozycyjnych w arytmetyce 'Float64' w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$. Należy również zbadać, jak liczby te są rozmieszczone w przedziałach $[1/2, 1]$ oraz $[2, 4]$.

3.2 Rozwiązanie i Wyniki

Analiza rozmieszczenia liczb z wykorzystaniem funkcji 'bitstring'. Te zadanie wyjątkowo wykonałem eksperymentując w terminalu Julii, jako jedyne nie ma swojego pliku .jd w pakiecie .zip z kodami źródłowymi

- **Przedział [1, 2]:** W pierwszej kolejności przypisałem do x1 wartość nexfloat(Float64(1)) oraz prevfloat(Float64(2)) do x2, po czym obejrzałem oba w 'bitstring' jak również 1 i 2. Dodatkowo przejrzałem co zwracają kolejne nextfloat'y.

  - bitstring(Float64(1)) =
    0 | 01111111111 | 0000000000000000000000000000000000000000000000000000
  - bitstring(x1) =
    0 | 01111111111 | 0000000000000000000000000000000000000000000000000001
  - bitstring(x2) =
    0 | 01111111111 | 1111111111111111111111111111111111111111111111111111
  - bitstring(Float64(2)) =
    0 | 10000000000 | 0000000000000000000000000000000000000000000000000000

  Jak widać do 2 cecha jest stała, a kolejne liczby z początku i końca tego przedziału różnią się o dokładnie jeden bit mantysy, czyli w arytmetyce *double* (przy stałej cesze, którą możemy odczytać jako $01111111111_2 = 1023_{10} = 0 + 1023$) o $2^0 * 2^{-52}$. Dodatkowo sprawdziłem dokładną potęgę 2 w rożnicy między liczbami w Julii: $log2(nextfloat(one(Float64)) - one(Float64)) = -52.0$

- **Przedział [1/2, 1]:** Zacząłem zanalogiczne do poprzedniego przedziału:
  - bitstring(Float64(1/2)) =
    0 | 01111111110 | 0000000000000000000000000000000000000000000000000000
  - bitstring(x1) =
    0 | 01111111110 | 0000000000000000000000000000000000000000000000000001
  - bitstring(x2) =
    0 | 01111111110 | 1111111111111111111111111111111111111111111111111111
  - bitstring(Float64(1)) =
    0 | 01111111111 | 0000000000000000000000000000000000000000000000000000

  Możemy zauważyć, że zmiany wyglądają bardzo podobnie, również o jeden bit 52-bitowej mantysy, lecz tym razem stała cecha jest inna a co za tym idzie - krok: $01111111110_2 = 1022_{10} = -1 + 1023- > \delta = 2^{-1} * 2^{-52} = 2^{-53}$. Sprawdzenie w Julii jak poprzednio: $log2(nextfloat(one(Float64)/2) - one(Float64)/2) = -53.0$.

- **Przedział [2, 4]:** W tym przedziale analogicznie jak poprzednio.

-   - bitstring(Float64(2)) =
      0 | 10000000000 | 0000000000000000000000000000000000000000000000000000
    - bitstring(x1) =
      0 | 10000000000 | 0000000000000000000000000000000000000000000000000001
    - bitstring(x2) =
      0 | 10000000000 | 1111111111111111111111111111111111111111111111111111
    - bitstring(Float64(4)) =
      0 | 10000000001 | 0000000000000000000000000000000000000000000000000000

  Analiza kroku: $10000000000_2 = 1024_{10} = +1 + 1023- > \delta = 2^1 * 2^{-52} = 2^{-51}$.
  Sprawdzenie w Julii jak wyżej: $log2(nextfloat(one(Float64) * 2) - one(Float64) * 2) = -51.0$

3.3 Wnioski

Kroki między kolejnymi liczbami są mniejsze im bliżej zera, co skutkuje więszką precyją. Natomiast im dalej od zera kroki są większe, więc precyzja mniejsza. Co więcej, można stwierdzić, że własność zauważona w zadaniu zachodzi dla wszystkich przedziałów [a, b], gdzie a < b oraz a i b są kolejnymi potęgami dwójki. Wynika to z faktu, że liczby z takiego przedziału w arytmetyce

Float64 mają ten sam wykładnik — różnią się jedynie mantysą. Wyjątkiem jest prawy koniec b: ma wykładnik większy o 1 i mantysę równą zero, podczas gdy jego lewy sąsiad ma mantysę zapełnioną jedynkami.

4 Zadanie 4: Błędy zaokrągleń

4.1 Opis problemu

a) znaleźć w arytmetyce ‘Float64‘ liczbę x z przedziału $(1, 2)$, dla której operacja $x \cdot (1/x)$ nie jest równa 1.
b) znaleźć najmniejszą taką liczbę.

4.2 Rozwiązanie

Napisałem funkcję, która do jedynki w arytmetyce *double* i "przestawia"na *nextfloat()* dopóki $x \cdot (1/x) == 1$, po czym zwraca te liczbę. To była też najmniejsza liczba z podanego przedziału, więc odpowiada też 'b)', więc napisałem jeszcze jedną funcję, która odejmuje *prevfloat()* od 2, żeby pokazać, że najmniejsza nie jest odosobnionym przypadkiem.

4.3 Wyniki

- Znaleziona liczba x: 1.9999999850988384,
- $x \cdot (1/x) = 0.9999999999999999$,
- Najmniejsza znaleziona liczba y: 1.000000057228997
- $y \cdot (1/y) = 0.9999999999999999$.

4.4 Wnioski

Dlaczego $fl(x \cdot fl(1/x)) \neq 1$? Ponieważ obliczenia obarczone są błędem, który należy brać pod uwagę nawet przy najprostrzych obliczeniach.

5 Zadanie 5: Iloczyn skalarny

5.1 Opis problemu

Celem zadania jest obliczenie iloczynu skalarnego dwóch wektorów na cztery różne sposoby (w przód, w tył, od największego do najmniejszego, od najmniejszego do największego) dla pojedynczej i podwójnej precyzji. Wyniki należy porównać z wartością referencyjną.

x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]
y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]

5.2 Rozwiązanie

Zaimplementowałem 4 algorytmy jak podane w zadaniu. Każdy operuje na arytmetyce podanych do niech argumentów. Na początku inicjalizuję x i y w *double* oraz tworzę zmienne x32 i y32 będące odpowiednio x i y w arytmetyce *single*. Następnie wywołuję funkcje po kolei i wypisuję na standardowe wyjście.

5.3 Wyniki

Wartość referencyjna: $-1.00657107000000 \cdot 10^{-11}$.

Tabela 5: Wyniki obliczeń iloczynu skalarnego.

Metoda	Float32	Float64
(a) "w przód"	-0.4999443	1.0251881368296672e-10
(b) "w tył"	-0.4543457	-1.5643308870494366e-10
(c) od największego	-0.5	0.0
(d) od najmniejszego	-0.5	0.0

5.4 Wnioski

Kolejność sumowania ma znaczenie. Wielkość błędu różni się dla każdego algorytmu. Żaden algorytm nie obliczył poprawnego wyniku. Obliczenie iloczynu skalarnego obarczone jest generalnie dużym błędem

6 Zadanie 6: Utrata precyzji

6.1 Opis problemu

Zadanie polega na obliczeniu wartości dwóch matematycznie równoważnych funkcji $f(x) = \sqrt{x^2+1} - 1$ oraz $g(x) = x^2/(\sqrt{x^2+1}+1)$ dla kolejnych wartości $x = 8^{-1}, 8^{-2}, \ldots$ i porównaniu wyników.

6.2 Rozwiązanie

Zaimplementowałem obie funkcje jak podano w zadaniu. Napisałem również funcję test, przyjmującą za argumenty x (w naszym przypadku 8) oraz n, gdzie n to liczba kolejnych porównań dla coraz mniejszych potęg. Funkcja ta wypisuje na standardowe wyjście wyniki obu opercji oraz ich różnicę w każdej iteracji. W tabeli poniżej znajdują się istotniejsze wyniki z wywołania funkcji test dla $x = 8.0$ oraz $n = 200.0$

6.3 Wyniki

Tabela 6: Porównanie $f(x)$ i $g(x)$

x	$f(x)$	$g(x)$	$\lvert f(x) - g()\rvert$
8^{-1}	0.0077822185373186414	0.0077822185373187065	6.505213034913027e-17
8^{-2}	0.00012206286282867573	0.00012206286282875901	8.328027937404281e-17
8^{-3}	1.9073468138230965e-6	1.907346813826566e-6	3.469446951953614e-18
8^{-4}	2.9802321943606103e-8	2.9802321943606116e-8	1.3234889800848443e-23
8^{-5}	4.656612873077393e-10	4.6566128719931904e-10	1.0842021724855044e-19
8^{-6}	7.275957614183426e-12	7.275957614156956e-12	2.6469779601696886e-23
8^{-7}	1.1368683772161603e-13	1.1368683772160957e-13	6.462348535570529e-27
8^{-8}	1.7763568394002505e-15	1.7763568394002489e-15	1.5777218104420236e-30
8^{-9}	0.0	2.7755575615628914e-17	2.7755575615628914e-17
8^{-10}	0.0	4.336808689942018e-19	4.336808689942018e-19
...	...	...	...
8^{-175}	0.0	4.144523e-317	4.144523e-317
8^{-176}	0.0	6.4758e-319	6.4758e-319
8^{-177}	0.0	1.012e-320	1.012e-320
8^{-178}	0.0	1.6e-322	1.6e-322
8^{-179}	0.0	0.0	0.0

6.4 Wnioski

Jak zaznaczono w zadaniu, z matematycznego punktu widzenia $f(x) = g(x)$. Dla pierwszych paru iteracji ($x \geq 8^{-8}$) wyniki funcji są zbliżone do siebie, a różnica między nimi jest (w zależności od poziomu precyzji, na jakiej nam zależy) pomijalnie mała. Jednak dla mniejszych liczb $f(x)$ przestało w ogóle zwracać wartość, a odczytywała 0.0. Można więc zauważyć, że funkcja $g(x)$ jest bardziej wiarygodna, jako iż lepiej zachowuje się dla stosunkowo małych x. Istotnym jest, że dla $x \to 0, \sqrt{x^2+1} \to 1$ oraz fakt, iż odejmowanie liczb w przybliżeniu równych obarczone jest błędem zależnym od argumentów działania. Jako, że w $g(x)$, dzięki przekształceniom matematycznym unikamy odejmowania, nie występuje taki problem jak w $f(x)$.

7 Zadanie 7: Różniczkowanie numeryczne

7.1 Opis problemu

Celem zadania jest obliczenie przybliżonej wartości pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x_0 = 1$ oraz błędów $|f'(x_0) - \tilde{f}'(x_0)|$ dla $h = 2^{-n}(n = 0, 1, 2, ..., 54)$. Wyniki należy zinterpretować. Należy również powiedzieć jak zachowują się wartości h + 1

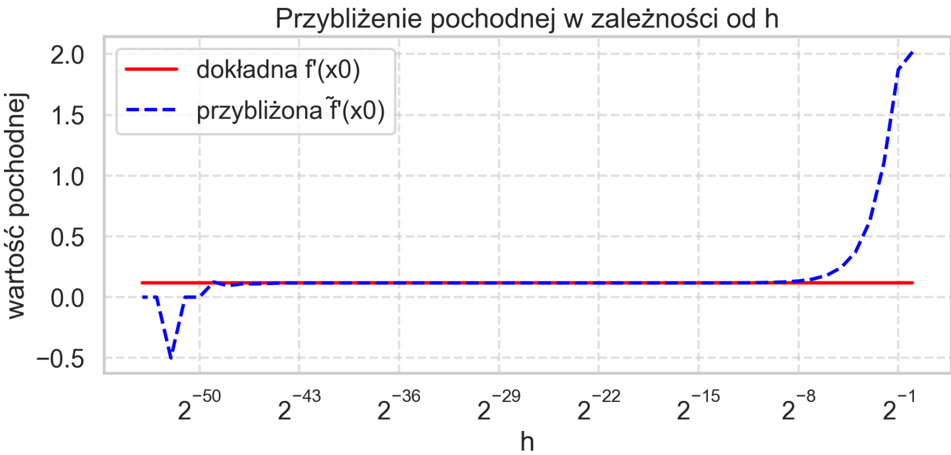
$$f'(x_0) \approx f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

7.2 Rozwiązanie

Zaimplementowałem w Julii funkcję z zadania, funkcję liczącą właściwą pochodną oraz funkcję aproksymującą pochodną wzorem podanym w zadaniu. Wywołuję je później w funkcji test, która dla podanych x i n (w tym przypadku 1 i 54 odpowiednio) spisuje wyniki dla $0 \leq i \leq n$ i zapisuje je do pliku .csv.

7.3 Wyniki

Poniższy wykres przedstawia błąd aproksymacji w funkcji kroku h.

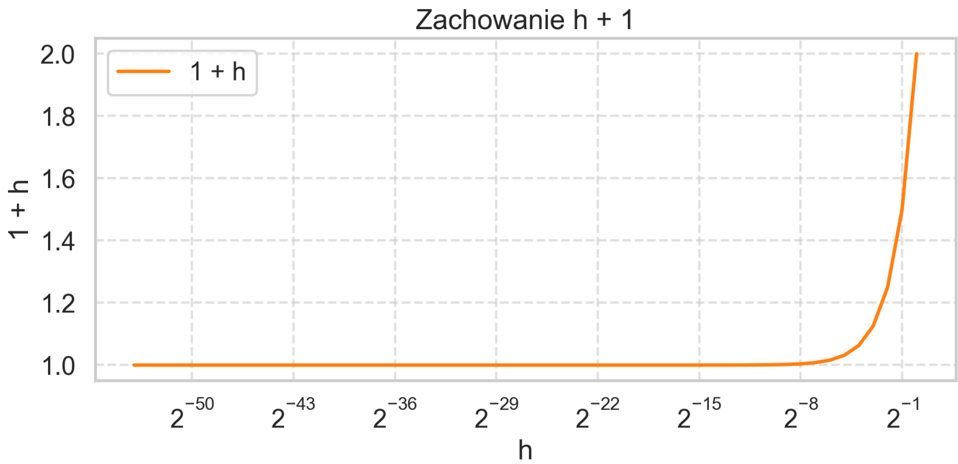


Rysunek 1: Wartość pochodnej

7.4 Interpretacja i Wnioski

Brak poprawy przybliżenia wartości pochodnej można wytłumaczyć tym, że dla odpowiednio małego h (jak możemy zaobserwować na Rysunku 2 dla $h < 2^{-28}$) $f(h+1) \approx f(1)$, a jak

Rysunek 2: Błąd obliczeń pochodnej.



Rysunek 3: Zachowanie h + 1.

wiemy m.in. z poprzedniego zadania odejmowanie liczb przybiżalnie małych wiąże się ze sporym błędem. W związku z czym jak h maleje, w arytmetyce "Float"przybliżenie w pewnym momencie przestaje działać zgodnie z modelem matematycznym, gdzie zmniejszanie h powoduje poprawę przybliżenia. Najlepsze przybiżenie w arytmetyce *double* otrzymałem dla $h = 2^{-28}$