Obliczenia naukowe

Lista 1

Aleksandra Czarniecka (272385)

28 października 2024

1 Rozpoznanie arytmetyki

Celem zadania jest zbadanie arytmetyki zmiennopozycyjnej w języku Julia poprzez wyznaczenie iteracyjnie trzech podstawowych wartości, takich jak: macheps, eta i max dla typów Float16, Float32 oraz Float64 zgodnych ze standardem IEEE 754. Wartości te będą porównywane z wynikami uzyskanymi przy użyciu wbudowanych funkcji w Julii (tj. eps(type), nextfloat(type(0.0)), floatmax(type)) oraz z wartościami zawartymi w nagłówku float.h języka C.

1.1 Epsilon maszynowy (macheps)

1.1.1 Opis problemu

Epsilon maszynowy to najmniejsza liczba nieujemna `macheps` taka, że $1 +$ `macheps` > 1. Jest to zatem odległość od 1 do najmniejszej liczby większej od 1 możliwej do zapisania w danej arytmetyce.

1.1.2 Rozwiązanie

Wyznaczanie epsilon maszynowego realizowane jest za pomocą funkcji, która inicjalizuje zmienną `macheps` wartością 1. W następnej kolejności, w ramach pętli, `macheps` jest dzielony przez 2, dopóki $1 +$ `macheps` > 1. Gdy warunek nie jest spełniony, zwrócony zostaje $2 \cdot$ `macheps`, czyli ostatnia wartość spełniająca nierówność.
W systemie binarnym operacja ta odpowiada przesunięciu jedynki w mantysie w prawo.

1.1.3 Wyniki oraz ich interpretacja

Typ zmiennopozycyjny	Wyznaczona wartość	Wartość eps(type)	Wartość z pliku float.h
Float16	0.000977	0.000977	-
Float32	1.1920929e-7	1.1920929e-7	1.192092896e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.220446049e-16

Tabela 1: Porównanie wartości epsilon maszynowego.

Wyniki wartości wyznaczanej iteracyjnie pokrywają się z wynikami funkcji eps(type), a także są bardzo zbliżone do wartości z pakietu float.h.

1.1.4 Związek epsilona maszynowego z precyzją arytmetyki

Precyzją arytmetyki oznaczaliśmy liczbę ϵ taką, że $\epsilon = 0.5 \cdot \beta^{1-t}$, gdzie β to wartość podstawy arytmetyki, czyli w tym przypadku $\beta = 2$, a t to liczba cyfr w mantysie.

Typ zmiennopozycyjny	Precyzja arytmetyki ϵ
Float16	$2^{-11} = 4.8828125000 \cdot 10^{-4}$
Float32	$2^{-24} = 5.9604644775390625 \cdot 10^{-8}$
Float64	$2^{-53} = 1.1102230246251565404236316680908203125 \cdot 10^{-16}$

Tabela 2: Precyzja arytmetyki dla typów zmiennopozycyjnych.

1.1.5 Wnioski

Program poprawnie liczy epsilon maszynowy dla wszystkich testowanych typów zmiennopozycyjnych.
Epsilon maszynowy jest powiązany z precyzją arytmetyki. Na podstawie Tabeli 1 oraz Tabeli 2 można wywnioskować, że: $macheps = 2 \cdot \epsilon$, zatem precyzja arytmetyki jest 2 razy mniejsza niż epsilon maszynowy. Wynika to z tego, że precyzja arytmetyki to maksymalna możliwa różnica między dokładną wartością, a jej przybliżeniem w arytmetyce. Zatem precyzja arytmetyki jest równa połowie odległości między 2 sąsiednimi liczbami (przybliżenie do najbliższej liczby), co określa już *macheps*.

1.2 Najmniejsza liczba maszynowa większa od 0 (eta)

1.2.1 Opis problemu

Liczba maszynowa eta to najmniejsza liczba η taka, że $\eta > 0$. Jest to zatem najmniejsza liczba większa od 0 w danej arytmetyce.

1.2.2 Rozwiązanie

Wyznaczanie liczby maszynowej eta realizowane jest za pomocą funkcji, która inicjalizuje zmienną `eta` wartością 1. W następnej kolejności, w ramach pętli, `eta` jest dzielona przez 2, aż do momentu, gdy wartość $\frac{\text{eta}}{2}$ stanie się mniejsza lub równa 0. Wtedy zwracana jest `eta`.

1.2.3 Wyniki oraz ich interpretacja

Typ zmiennopozycyjny	Wyznaczona wartość	Wartość nextfloat(type(0.0))
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Tabela 3: Porównanie wartości liczby maszynowej eta.

Wyniki wartości wyznaczanej iteracyjnie pokrywają się z wynikami funkcji nextfloat(type(0.0)).

1.2.4 Związek liczby maszynowej eta z liczbą MIN_{sub}

Liczba MIN_{sub} to najmniejsza możliwa liczba subnormalna, którą można zapisać w standardzie IEEE 754. Możemy ją obliczyć za pomocą wzoru:

$$\text{MIN}_{\text{sub}} = 2^{-(t\ -\ 1)} \cdot 2^{c_{\text{min}}}$$

,gdzie t- liczba cyfr w mantycie, c_{min}- minimalna wartość cechy *c*.

Typ zmiennopozycyjny	MIN_{sub}
Float16	$2^{-11-1} \cdot 2^{-14} = 5.9604644775390625 \cdot 10^{-8}$
Float32	$2^{-24-1} \cdot 2^{-126} = 1.401298464324817 \cdot 10^{-45}$
Float64	$2^{-53-1} \cdot 2^{-1022} = 4.94065645841246 \cdot 10^{-324}$

Tabela 4: Wartości MIN_{sub} dla typów zmiennopozycyjnych.

Analizując Tabele 3 oraz Tabele 4 można zauważyć, że watości MIN_{sub} i *eta* są do siebie zbliżone.

1.2.5 Funkcje floatmin(type), a liczba MIN_{nor}

Funkcja wbudowana w Julii floatmin(type) zwraca wartość najmniejszej dodatniej liczby znormalizowanej, którą można zapisać w standardzie IEEE 754. Liczba MIN_{nor} to najmniejsza możliwa liczba znormalizowana, którą można zapisać w standardzie IEEE 754, którą można zapisać w standardzie IEEE 754. Możemy ją obliczyć za pomocą wzoru:

$$\text{MIN}_{\text{nor}} = 2^{c_{\text{min}}}$$

,gdzie c_{min}- minimalna wartość cechy c.

Typ zmiennopozycyjny	MIN_{nor}	Wartość floatmin(type)
Float16	-	-
Float32	$2^{-126} = 1.175494350822287507968 \cdot 10^{-38}$	1.1754944e-38
Float64	$2^{-1022} = 2.225073858507201383 \cdot 10^{-308}$	2.2250738585072014e-308

Tabela 5: Porównanie wartości MIN_{NOR} i funkcji wbudowanej floatmin(type) dla typów zmiennopozycyjnych.

Można zauważyć, że watości MIN_{nor} i wartości zwracane przez funkcję floatmin(type) są do siebie bardzo zbliżone.

1.2.6 Wnioski

Program poprawnie liczy liczbę maszynową eta dla wszystkich testowanych typów zmiennopozycyjnych.
Eta to najmniejsza liczba, którą można zapisać w standardzie IEEE 754 i ma wartość zbliżoną do watości MIN_{sub}.

1.3 Liczba (MAX)

1.3.1 Opis problemu

Liczba (MAX) to największa możliwa do wyrażenia liczba dla danej arytmetyki, zatem jej cecha ma maksymalną dopuszczalną wartość, a mantysa składa się z samych jedynek.

1.3.2 Rozwiązanie

W systemie binarnym dzielenie przez 2 można zrozumieć jako przesunięcie przecinka w liczbie o jedno miejsce w lewo. W efekcie tej operacji mantysa pozostaje bez zmian, a zmienia się jedynie wykładnik CC. W standardzie IEEE 754, opartym na notacji wykładniczej o podstawie 2, celem jest odnalezienie największej liczby, która może być reprezentowana, mającej maksymalną liczbę jedynek w części ułamkowej mantysy, czyli 1.111...1.111.... Następnie liczba ta jest mnożona przez 2, aby uzyskać ostateczną wartość.

Wyznaczanie największej liczby maszynowej realizowane jest za pomocą funkcji, która najpierw inicjalizuje zmienną `prev_max` oraz zmienną `max` wartością 1. Następnie w pętli wykonywana jest operacja, która dzieli `prev_max` przez 2, dopóki różnica pomiędzy `max` a `prev_max` jest mniejsza niż 1. Po zakończeniu pierwszej pętli, wartość `max` jest aktualizowana poprzez odjęcie podwójnej wartości `prev_max`. Za pomocą funkcji isinf sprawdzamy, czy kolejna inkrementacja nie spowoduje utworzenia liczby spoza dostępnego zakresu. Kolejnie, w drugiej pętli, następuje mnożenie `max` przez 2, aż do momentu, gdy wartość `max`·2 osiągnie nieskończoność. Ostatecznie funkcja zwraca obliczoną wartość `max`, reprezentującą największą liczbę maszynową.

1.3.3 Wyniki oraz ich interpretacja

Typ zmiennopozycyjny	Wyznaczona wartość	Wartość maxfloat(type)	Wartość z pliku float.h
Float16	6.55e4	6.55e4	-
Float32	3.4028235e38	3.4028235e38	3.402823e+38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.797693e+308

Tabela 6: Porównanie wartości liczby (MAX).

Wyniki wartości wyznaczanej iteracyjnie pokrywają się z wynikami funkcji maxfloat(type), a także są zbliżone do wartości z pakietu float.h.

1.3.4 Wnioski

Program poprawnie liczy liczbę (MAX) dla wszystkich testowanych typów zmiennopozycyjnych.

1.4 Wnioski ogólne

Reprezentacja liczb zmiennopozycyjnych w standardzie IEEE-754 posiada pewne ograniczenia, które wpływają na dokładność odwzorowania wartości liczbowych. Zwiększając liczbę bitów przeznaczonych na zapis liczby, te ograniczenia stają się mniej zauważalne. W związku z tym dokładność wyników jest lepsza przy użyciu arytmetyki o większej precyzji.

2 Wzór Kahana na epsilon maszynowy

2.1 Opis problemu

Celem zadania jest eksperymentalne sprawdzenie w języku Julia, czy można otrzymać epsilon maszynowy (macheps) obliczając wyrażenie Kahana $3 \cdot (\frac{4}{3} - 1)$ - 1 w arytmetyce zmiennopozycyjnej dla wszystkich typów zmiennopozycyjnych Float16, Float32, Float64.

2.2 Rozwiązanie

Rozwiązanie polega na zaimplementowaniu funkcji, która dla danego typu zmiannopozycyjnego obliczy wyrażenie. Następnie należy sprawdzić poprawność wyników porównując wyrażenia Kahana z wynikami wbudowanej funkcji eps(type).

2.3 Wyniki oraz ich interpretacja

Typ zmiennopozycyjny	Wartość wyrażenia Kahana	Wartość eps(type)
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Tabela 7: Porównanie wartości epsilon maszynowego.

Z tabeli wynika, że wyniki mają tę samą wartość bezwzględną. Różnice występują w Float16 i Float64, gdzie zmienia się znak. Dzieje się tak z powodu zasady „round to even", która zaokrągla wyniki do najbliższej liczby parzystej. Gdy wynik jest bliski wartości pośredniej, może to prowadzić do zaokrąglenia w górę lub w dół, co w niektórych przypadkach zmienia znak wyniku.

2.4 Wnioski

Wyrażenie Kahana poprawnie liczy epsilon maszynowy co do wartości bezwzględnej. Należy jednak pamiętać, że mogą wystąpić pewne problemy związane z problemem przybliżeń góra/dół.

3 Równomierne rozmieszczenie liczb w przedziale

3.1 Opis problemu

Celem zadania jest eksperymentalne sprawdzenie, czy w arytmetyce Float64 zgodnej ze standardem IEEE 754 (double) liczby w przedziale [1, 2] są równomiernie rozłożone z krokiem $\delta = 2^{-52}$. Oznacza to, że każda liczba zmiennopozycyjna x pomiędzy 1 i 2 może być przedstawione następująco $x = 1 + k \cdot \delta$ w tej arytmetyce, gdzie $k = 1, 2, ..., 2^{52} - 1$ i $\delta = 2^{-52}$. Następnie sprawdzenie zależności także dla przedziałów: $[\frac{1}{2}, 1]$, $[2, 4]$.

3.2 Rozwiązanie

Roziązanie powinno sprawdzać wszystkie dostępne liczby, jednak byłoby to bardzo kosztowne. Zatem obserwacje będą musiały zostać uproszczone. Pomocna będzie wbudowana funkcja języka Julia bitstring(), dzięki której można przyjrzeć się zapisowi bitowemu kolejnych liczb typu Float64. Program dla przedziału [left, right], oblicza i porównuje oczekiwaną wartość odstępu (dδ- delta) między liczbami zmiennoprzecinkowymi. δ jest wyznaczana na podstawie wykładnika liczby, co pozwala oszacować różnicę wynikającą z ograniczeń precyzji IEEE 754. Wynikowa δ jest następnie

porównywana z rzeczywistą różnicą pomiędzy liczbą left i jej kolejną liczbą maszynową. Oprócz tego program posiada funkcje, które służą do wyświetlenia binarnej reprezentacji kolejnych lub poprzednich liczb zmiennoprzecinkowych względem danej liczby x. Funkcje te są przydatne do obserwacji minimalnych przyrostów w liczbach zmiennoprzecinkowych.

3.3 Wyniki oraz ich interpretacja

Liczba	Zapis bitowy
1.0	0 01111111111 0000000000000000000000000000000000000000000000000000
$1.0 + \delta$	0 01111111111 0000000000000000000000000000000000000000000000000001
nextfloat(1.0)	0 01111111111 0000000000000000000000000000000000000000000000000001
$1.0 + (2^{52} - 1) \cdot \delta$	0 01111111111 1111111111111111111111111111111111111111111111111111
prevfloat(2.0)	0 01111111111 1111111111111111111111111111111111111111111111111111
2.0	0 10000000000 0000000000000000000000000000000000000000000000000000

Tabela 8: Zapisy bitowe liczb w przedziale $[1, 2]$

Liczba	Zapis bitowy
0.5	0 01111111110 0000000000000000000000000000000000000000000000000000
$0.5 + \delta$	0 01111111110 0000000000000000000000000000000000000000000000000001
nextfloat(0.5)	0 01111111110 0000000000000000000000000000000000000000000000000001
$0.5 + (2^{52} - 1) \cdot \delta$	0 01111111110 1111111111111111111111111111111111111111111111111111
prevfloat(1.0)	0 01111111110 1111111111111111111111111111111111111111111111111111
1.0	0 01111111111 0000000000000000000000000000000000000000000000000000

Tabela 9: Zapisy bitowe liczb w przedziale $[0.5, 1]$

Liczba	Zapis bitowy
2.0	0 10000000000 0000000000000000000000000000000000000000000000000000
$2.0 + \delta$	0 10000000000 0000000000000000000000000000000000000000000000000001
nextfloat(2.0)	0 10000000000 0000000000000000000000000000000000000000000000000001
$2.0 + (2^{52} - 1) \cdot \delta$	0 10000000000 1111111111111111111111111111111111111111111111111111
prevfloat(4.0)	0 10000000000 1111111111111111111111111111111111111111111111111111
4.0	0 10000000001 0000000000000000000000000000000000000000000000000000

Tabela 10: Zapisy bitowe liczb w przedziale $[2, 4]$

Z tabel można odczytać, że obliczone wartości za pomocą δ pokrywając się z wywoływanymi funkcjami wbudowanymi języka Julii.

Delta w przedziale [1, 2] wynosi $\delta = 2^{-52} = 2.220446049250313e - 16$.
Delta w przedziale [0.5, 1] wynosi $\delta = 2^{-53} = 1.1102230246251565e - 16$.
Delta w przedziale [2, 4] wynosi $\delta = 2^{-51} = 4.440892098500626e - 16$.

3.4 Wnioski

Liczby znajdujące się pomiędzy kolejnymi potęgami liczby 2 są rozmieszczone równomiernie. Każdą liczbę z przedziału $[2^t, 2^{t+1}]$ (t- liczba całkowita) można zapisać jako $2^t + k \cdot \delta$, gdzie $k \in 1, 2, \ldots, 2^{52} - 1$, a $\delta = 2^{-52+t}$.

4 Poszukiwanie liczby zmiennopozycyjnej x w przedziale $1 < x < 2$

4.1 Opis problemu

Celem zadania jest eksperymentalne znalezienie w arytmetyce Float64 zgodnej ze standardem IEEE 754 (double) najmniejszej liczby zmiennopozycyjnej x w przedziale $1 < x < 2$, dla której zachodzi nierówność $x \cdot \left(\frac{1}{x}\right) \neq 1$.

4.2 Rozwiązanie

Wyznaczanie x realizowane jest za pomocą funkcji, która inicjalizuje zmienną `x` wartością 1, a następnie definiowana jest zmienna `d` jako 2.0^{-52}, co odpowiada różnicy między kolejnymi liczbami zmiennopozycyhnymi w tym przedziale (wiemy to z zadania 3). Funkcja wykonuje pętlę, która trwa do momentu, gdy x jest mniejsze od 2. W każdej iteracji sprawdzany jest warunek: jeśli $x \cdot \left(\frac{1}{x}\right) \neq 1$, funkcja zwraca wartość `x`. W przeciwnym razie, `x` jest zwiększane o `d`. Jeśli pętla zakończy się bez znalezienia odpowiedniego `x`, funkcja zwraca `nothing`. Na koniec program wyświetla najmniejszą liczbę `x`, dla której nierówność jest spełniona.

4.3 Wyniki oraz ich interpretacja

Program znajduje taką liczbę i zwraca ją: 1.000000057228997. Jest to najmniejsza liczba, ale nie jedyna, dla której tożsamość algebraiczna przestaje być prawdziwa.

4.4 Wnioski

Działania arytmetyczne (a zwłaszcza dzielenie, które jest nieodwracalne w tej arytmetyce) na liczbach zmiennopozycyjnych mogą generować błędy z powodu ograniczeń dokładności w arytmetyce zmiennopozycyjnej, a dokładniej problemów z zaokrąglaniem. Nawet tożsamości algebraiczne mogą prowadzić do nieoczekiwanych wyników, takich jak: $x \cdot \left(\frac{1}{x}\right) \neq 1$.

5 Obliczanie iloczynu skalarnego dwóch wektorów

5.1 Opis problemu

Celem zadania jest eksperymentalne obliczenie iloczynu skalarnego dwóch wektorów:

$$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$

$$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049].$$

W związku z tym należy zaimplementować 4 różne algorytmy sumowania dla n = 5 używając pojedynczej i podwójnej precyzji (typy Float32 i Float64 w języku Julia). Następnie porównanie wyników z prawidłową wartością (dokładność do 15 cyfr) $-1.00657107000000 \cdot 2.0^{-52}$.

5.2 Rozwiązanie

Rozwiązanie polega na zaimplementowaniu w Julii 4 algorymów:

- "w przód"a_i od $i = 1$ do n jest dana przez $\sum_{i=1}^{n} x_i y_i$
- "w tył"a_i od $i = n$ do 1 jest dana przez $\sum_{i=1}^{n} x_i y_i$
- od największego do najmniejszego (dodaj dodatnie liczby w porządku od największego do najmniejszego, dodaj ujemne liczby w porządku od najmniejszego do największego, a następnie daj do siebie obliczone sumy częściowe)
- od najmniejszego do największego (przeciwnie do metody poprzedniej)

5.3 Wyniki oraz ich interpretacja

Typ	"w przód"	"w tył"	od najw. do najmn.	od najmn. do najw.
Float32	-0.4999443	-0.4543457	-0.5	-0.5
Float64	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0

Tabela 11: Porównanie wyników iloczynu skalarnego wektorów.

Z tabeli wynika, że wynik iloczynu skalarnego zależy od rodzaju algorytmu. Dla większości sposobów wyniki się od siebie różnią. Prawidłowa wartość (dokładność do 15 cyfr) wynosi: $-1.00657107000000 \cdot 2.0^{-52}$, co także nie pokrywa się z otrzymanymi wynikami. Można jednak zauważyć, że arytmetyka Float64 była dokładniejsza niż arytmetyka Float32.

5.4 Wnioski

Kolejność wykonywania działań ma istotny wpływ na wyniki obliczeń numerycznych. Zjawisko to jest widoczne w zadaniu w kontekście operacji dodawania, gdzie suma dużej liczby i małej może prowadzić do znacznych błędów zaokrągleń. W takich przypadkach mniejszy składnik może być zdominowany przez większy, co skutkuje utratą precyzji w wyniku końcowym. Dokładność wyników jest lepsza przy użyciu arytmetyki o większej precyzji.

6 Analiza równości funkcji w arytmetyce zmiennopozycyjnej

6.1 Opis problemu

Celem zadania jest policzenie wartości funkcji w arytmetyce Float64:

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

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

dla kolejnych wartości argumentu x = 8^{-i}, gdzie i = 1, 2, 3, ...
Następnie należy przeanalizować otrzymane wyniki, pamiętając, że matematycznie f = g.

6.2 Rozwiązanie

Rozwiązanie polega na zaimlementowaniu dwóch funkcji oraz obliczeniu i przeanalizowaniu ich wartości dla x = 8^{-i}, dla początkowych i.

6.3 Wyniki oraz ich interpretacja

x	f(x)	g(x)
8^{-1}	0.0077822185373186414	0.0077822185373186414
8^{-2}	0.00012206286282867573	0.00012206286282867573
8^{-3}	1.9073468138230965e-6	1.9073468138230965e-6
...	...	...
8^{-8}	1.7763568394002505e-15	1.7763568394002505e-15
8^{-9}	0.0	2.7755575615628914e-17
...	...	...
8^{-178}	0.0	1.6e-322
8^{-179}	0.0	0.0

Tabela 12: Porównanie wartości funkcji.

Do x = 8^{-8} wyniki wartości obu funkcji się pokrywają. Następnie dla x = 8^{-9} wartość funkcji f(x) zeruje się, a funkcja g(x) zeruje się dopiero przy x = 8^{-179}.

6.4 Wnioski

Analizując wyniki, można stwierdzić, że funkcja f(x) jest mniej dokładana niż funkcja g(x). Jest to spodowane tym, że w funkcji f(x) dla x dążącego do 0, odejmujemy bliskie sobie liczby, co prowadzi do zmniejszenia precyzji arytmetyki i funkcja szybko zaczyna się zerować. Natomiast funkcja g(x) lepiej unika takiej sytuacji. W związku z tym warto zapisywać wyrażenia w lepszej do obliczeń formie.

7 Pochodna funkcji w punkcie x_0 i jej przybliżona wartość

7.1 Opis problemu

Celem zadania jest analiza błędu bezwzględnego pomiędzy wartością dokładną pochodnej funkcji $f(x) = sin(x) + cos(3x)$ w punkcie $x_0 = 1$, a wartością przybliżenia za pomocą wzoru:

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

dla h = 2^{-n}, gdzie n = 0, 1, 2, ..., 54.

7.2 Rozwiązanie

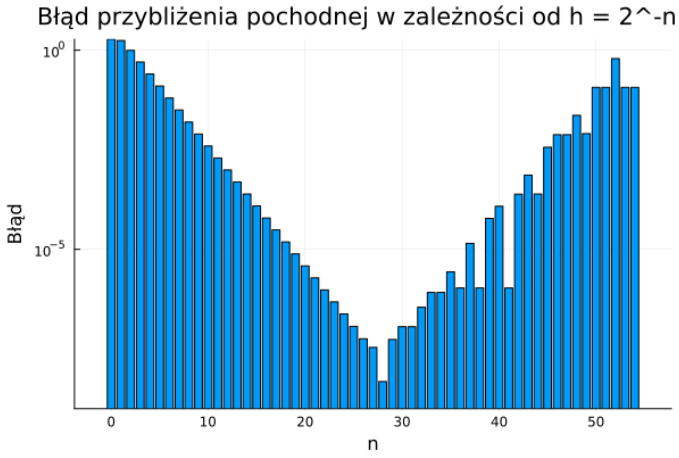
Rozwiązanie polega na zaimpementowaniu powyższych funkcji, obliczeniu nimi dokładnej i przybliżonej wartości pochodnej funkcji w punkcie $x_0 = 1$. Następnie obliczeniu i analizie błędu danego wzorem $\left|f'(x_0) - \tilde{f}'(x_0)\right|$ dla h = 2^{-n}, gdzie n = 0, 1, 2, ..., 54.

7.3 Wyniki oraz ich interpretacja

h	h + 1	$\tilde{f}'(x_0)$	$\left\|f'(x_0) - \tilde{f}'(x_0)\right\|$
2^{-0}	2.0	2.0179892252685967	1.9010469435800585
2^{-1}	1.5	1.8704413979316472	1.753499116243109
2^{-2}	1.25	1.1077870952342974	0.9908448135457593
...	...	...	...
2^{-25}	1.0000000298023224	0.116942398250103	1.1656156484463054e-7
2^{-26}	1.0000000149011612	0.11694233864545822	5.6956920069239914e-8
...	...	...	...
2^{-53}	1.0	0.0	0.11694228168853815
2^{-54}	1.0	0.0	0.11694228168853815

Tabela 13: Porównanie wartości epsilon maszynowego.

Dokładna wartość pochodnej funkcji wynosi: 0.11694228168853815.
Wartość 1 + h wraz ze wzrostem samego h zbliża się do 1.0.



Rysunek 1: Błąd względny przybliżenia pochodnej funkcji.

Wykres ilustruje, jak w zależności od wartości h, rozkłada się błąd bezwzględny przybliżenia pochodnej w punkcie. Dla $h = 2^{-28}$ błąd jest najmniejszy, dla tej wartości h uzyskujemy najlepsze przybliżenie, natomiast $h = 2^{-0}$ najgorsze.

7.4 Wnioski

Zmniejszanie h poprawia wynik przybliżenia jedynie do pewnego momentu. Następnie błąd znowu powiększa się, ponieważ przy operacjach na liczbach bliskich sobie dokładność obliczeń jest ograniczona ze względu na redukcję cyfr znaczących. Dla coraz mniejszych h będziemy tracić coraz więcej cyfr znaczących i odwrotnie niż w matematyce, czym mniejsze h tym gorsze przybliżenie.

Dalsze zmniejszanie h nie poprawi przybliżenia, ponieważ dla wartości $h \leq 2^{-53}$ występuje równość 1 + h = h w arytmetyce zmiennopozycyjnej. Wynika to z ograniczonej precyzji reprezentacji liczb, gdzie przy bardzo małych wartościach h dodanie tej liczby do 1 nie powoduje zauważalnej zmiany — wartość h zostaje zignorowana podczas zaokrąglania.