Obliczenia naukowe - lab1

Jakub Musiał 268442

Październik 2023

Zadanie 1.1 - Epsilon Maszunowy

Problem

Napisać program w języku `Julia` wyznaczający iteracyjnie wartości ϵ_{mach} takich, że:

$$\begin{cases} \epsilon_{mach} > 0 \\ fl(1.0 + \epsilon_{mach}) > 1.0 \\ fl(1.0 + \epsilon_{mach}) = 1 + \epsilon_{mach} \end{cases} \tag{1}$$

dla wszystkich typów zmiennopozycyjnych: `Float16`, `Float32`, `Float64`.
Można zauważyć, że ϵ_{mach} jest odległością od 1 najmniejszej w zadanej arytmetyce liczby, której dodanie do 1 skutkuje zmianą przechowywanej wartości.

Rozwiązanie

Liczbę maszynową ϵ_{mach} można otrzymać poprzez iteracyjne sprawdzanie kolejnych (co raz mniejszych) potęg liczby 2, zaczynając od $2^0 = 1$, dopóki ograniczenia (1) na wartość liczby ϵ_{mach} są spełnione.

Programy z rozwiązaniem:

- `ex1/mach_eps.jl`
- `ex1/mach_eps.c`

Wyniki i obserwacje

Na podstawie *tabeli 1* możemy zauważyć, że wyniki otrzymane iteracyjnie pokrywają się z tymi zwróconymi przez funkcję `eps`, jak i z tymi zdefiniowanymi w standardzie języka `C` (wyłącznie w precyzji pojedynczej i podwójnej, jako że w języku `C` nie ma odpowiednika `Float16`).

Precyzja	Obliczone ϵ_{mach}	`eps`	`float.h`
Float16	0.000977	0.000977	brak
Float32	$1.1920929e-7$	$1.1920929e-7$	$1.192093e-07$
Float64	$2.220446049250313e-16$	$2.220446049250313e-16$	$2.220446e-16$

Table 1: Wartości liczby ϵ_{mach} dla poszczególnych precyzji.

Związek liczby ϵ_{mach} z precyzją arytmetyki:
Precyzja arytmetyki ϵ jest określona wzorem:

$$\epsilon = \frac{1}{2}\beta^{1-t}$$

Przy czym β jest podstawą systemu, a t jest liczbą cyfr mantysy. Wiedząc, że $\beta = 2$, możemy przekształcić powyższy wzór:

$$\epsilon = \frac{1}{2} \cdot 2^{1-t} = 2^{-t}$$

Możemy zatem wyznaczyć wartości precyzji arytmetyk zmiennopozycyjnych.

Precyzja	t	$\epsilon = fl(2^{-t})$	$epsilon_{mach}$
Float16	10	0.000977	0.000977
Float32	23	$1.1920929e-7$	$1.1920929e-7$
Float64	52	$2.220446049250313e-16$	$2.220446049250313e-16$

Table 2: Porównanie wartości ϵ_{mach} oraz precyzji arytmetyki

Na podstawie powyższej tabeli możemy stwierdzić, że $\epsilon = \epsilon_{mach}$.

Zadanie 1.2 - Maszynowe eta

Problem

Napisać program w języku `Julia` wyznaczający iteracyjnie wartości $\eta_{mach} > 0$ dla wszystkich typów zmiennopozycyjnych: `Float16`, `Float32`, `Float64`.
Można zauważyć, że η_{mach} jest najmniejszą nieujemną liczbą, którą da się zapisać w zadanej precyzji (nie wpada w przedział zera maszynowego).

Rozwiązanie

Liczbę maszynową η_{mach} można otrzymać poprzez iteracyjne sprawdzanie kolejnych (co raz mniejszych) potęg liczby 2, zaczynając od $2^0 = 1$, dopóki wyliczona wartość $\eta_{mach} > 0$.

Program z rozwiązaniem: `ex1/mach_eta.jl`

Wyniki i obserwacje

Na podstawie *tabeli 3* możemy zauważyć, że wyniki otrzymane iteracyjnie pokrywają się z tymi zwróconymi przez funkcję wbudowaną `nextfloat(0.0)`

Precyzja	Obliczone η_{mach}	`nextfloat(0.0)`
Float16	$6.0e-8$	$6.0e-8$
Float32	$1.0e-45$	$1.0e-45$
Float64	$5.0e-324$	$5.0e-324$

Table 3: Wartości liczby η_{mach} dla poszczególnych precyzji.

Zauważmy, że liczba η_{mach} w zapisie bitowym ma następującą postać:

$$\eta_{mach}(precyzja) : 0 \ \underbrace{0...0}_{cecha} \ \underbrace{0...01}_{mantysa}$$

Jest to zatem liczba nieznormalizowana, ponieważ cecha liczby η_{mach} to same zera

Związek liczby η_{mach} z wartościa MIN_{sub}:
Wiemy, że MIN_{sub} jest namniejsza liczbą nieznormalizowaną w zadanej arytmetyce i jest określona wzorem:

$$MIN_{sub} = 2^{1-t} \cdot 2^{c_{min}} = 2^{1-t+c_{min}}$$

Zatem, wiedząc, że $c_{min} = -2^{d-1} + 2$ gdzie $d \equiv$ liczba bitów przeznaczonych na zapis cechy w standardzie `IEEE 754`, możemy wyznaczyć wartości minimalne w postaci nieznormalizowanej dla zadanych precyzji.

Precyzja	t	d	c_{min}	MIN_{sub}	η_{mach}
Float16	10	5	-14	$fl(2^{-24}) = 6.0e-8$	$6.0e-8$
Float32	23	8	-126	$fl(2^{-149}) = 1.0e-45$	$1.0e-45$
Float64	52	11	-1022	$fl(2^{-1074}) = 5.0e-324$	$5.0e-324$

Table 4: Porównanie wartości MIN_{sub} oraz η_{mach}

Na podstawie powyższej tabeli możemy stwierdzić, że $MIN_{sub} = \eta_{mach}$.

Związek wartości MIN_{nor} z funkcją `floatmin()`: Wiemy, że wartość MIN_{nor} możemy uzyscać ze wzoru:

$$MIN_{nor} = 2^{c_{min}}$$

Jak powyżej obliczamy $c_{min} = -2^{d-1} + 2$.
Zatem porównajmy zadany wzór z wartościami zwracanymi przez funkcję `floatmin()`. Z

Precyzja	d	c_{min}	MIN_{nor}	`floatmin`
Float16	5	-14	$6.104e-5$	$6.104e-5$
Float32	8	-126	$1.1754944e-38$	$1.1754944e-38$
Float64	11	-1074	$2.2250738585072014e-308$	$2.2250738585072014e-308$

Table 5: Porównanie wartości MIN_{nor} oraz funkcji `floatmin()`

powyższej tabeli wynika, że funkcja wbudowana `floatmin()` języka `Julia` zwraca wartości MIN_{nor} dla zadanej precyzji.

Zadanie 1.3 - Maximum float

Problem

Napisać program w języku `Julia` wyznaczający iteracyjnie wartości fl_{max} - największej liczby, którą da się zapisać w arytmetyce zmiennopozycyjnej - dla wszystkich typów zmiennopozycyjnych: `Float16`, `Float32`, `Float64`.

Rozwiązanie

Liczbę fl_{max} można otrzymać poprzez iteracyjne sprawdzanie kolejnych (co raz większych) potęg liczby 2, zaczynając od $2^0 = 1$, dopóki wyliczona wartość $fl_{max} < \infty$, a następnie dodając coraz mniejsze potęgi liczby 2, zaczynając od $2^{i_{max}-1}$, ponownie dopóki wyliczona wartość $fl_{max} < \infty$

Programy z rozwiązaniem:

- `ex1/max_float.jl`
- `ex1/max_float.c`

Wyniki i obserwacje

Na podstawie *tabeli 5* możemy zauważyć, że wyniki otrzymane iteracyjnie pokrywają się z tymi zwróconymi przez funkcję wbudowaną `nextfloat(0.0)`, jak i z tymi zdefiniowanymi w standardzie języka `C` (podobnie jak w zadaniu 1.1 wyłącznie w precyzji pojedynczej i podwójnej, jako że w języku `C` nie ma odpowiednika `Float16`).

Precyzja	Obliczone fl_{max}	`floatmax`	`float.h`
Float16	$6.55e4$	$6.55e4$	brak
Float32	$3.4028235e38$	$3.4028235e38$	$3.402823e+38$
Float64	$1.7976931348623157e308$	$1.7976931348623157e308$	$1.797693e+308$

Table 6: Wartości liczby fl_{max} dla poszczególnych precyzji.

Zauważmy, że liczba fl_{max} w zapisie bitowym ma następującą postać:

$$fl_{max}(precyzja) : 0 \underbrace{1...10}_{cecha} \underbrace{1...1}_{mantysa}$$

Zadanie 2 - Kahan machine epsilon

Problem

Sprawdzić poprawność wzoru Kahana na obliczanie wartości ϵ_{mach} dla wszystkich typów zmiennopozycyjnych: `Float16`, `Float32`, `Float64`.

Rozwiązanie

Wzór Kahana:

$$\epsilon_{kahan} = 3 \cdot (\tfrac{4}{3} - 1) - 1$$

Program z rozwiązaniem: `ex2.jl`

Wyniki i obserwacje

Precyzja	ϵ_{kahan}	ϵ_{mach}
Float16	-0.000977	0.000977
Float32	$1.1920929e-7$	$1.1920929e-7$
Float64	$-2.220446049250313e-16$	$2.220446049250313e-16$

Table 7: Wartości liczby ϵ_{kahan} dla poszczególnych precyzji.

Na podstawie wyników w *tabeli 7* możemy zauważyć, że wartości otrzymywane z wzoru Kahana są równe wartościom ϵ_{mach} co do wartości bezwzględnej, zatem, aby poprawić ten wzór należałoby nałożyć moduł na zwrócony wynik:

$$\epsilon_{correct\ kahan} = |3 \cdot (\tfrac{4}{3} - 1) - 1|$$

Zadanie 3 - Rozmieszczenie liczb w arytmetyce zmiennopozycyjnej w podwójnej precyzji

Problem

Sprawdzić w języku `Julia`, czy liczby są równomiernie rozmieszczone w arytmetyce `Float64` w przedziałach: $[\frac{1}{2}, 1]$, $[1, 2]$ oraz $[2, 4]$

Rozwiązanie

Sprawdzenie wartości $x = 1 + k \cdot \delta$ dla $k \, \epsilon \, \{1, 2, ..., 2^{52} - 1\}$

Program z rozwiązaniem: `ex3.jl`

Wyniki i obserwacje

Przedział $[1, 2]$

Dla zadanej wartości $\delta = 2^{-52}$ uzyskujemy poniższe wartości:

Wartość	Postać binarna (znak bitu, cecha, mantysa)
1.0000000000000002	0 01...1 0...001
1.0000000000000004	0 01...1 0...010
1.0000000000000007	0 01...1 0...011
⋮	⋮
1.9999999999999993	0 01...1 1...101
1.9999999999999996	0 01...1 1...110
1.9999999999999998	0 01...1 1...111

Table 8: Rozkład liczb podwójnej precyzji w przedziale $[1, 2]$

W *tabeli 8* możemy zauważyć, że dodanie $\delta = 2^{-52}$ do liczby x z przedziału $[1, 2]$ powoduje dodanie wartości binarnej 1 do m_x (mantysy liczby x). Zatem możemy stwierdzić, że liczby w tym przedziale są równomiernie rozmieszczone z krokiem 2^{-52}.

Przedział $[\frac{1}{2}, 1]$

Wiemy, że liczb w przedziale $[\frac{1}{2}, 1]$ jest tyle samo, co w przedziale $[1, 2]$, ponieważ liczby w zadanym przedziale $[2^i, 2^{i+1}]$ mają tę samą cechę, a możliwych wartości mantysy jest $2^t = 2^{52}$. Zauważmy także, że dla $i = j - 1$ ($i, j \equiv$ wartości cechy) liczby reprezentowane przez tę samą mantysę z cechą i są dwukrotnie mniejsze od tych z cechą j, czyli $x_i = 2^{-1} \cdot x_j$.
Stąd możemy założyć, że dla przedziału $[\frac{1}{2}, 1]$ powinniśmy przyjąć krok $\delta = 2^{-52-1} = 2^{-53}$

Wartość	Postać binarna (znak bitu, cecha, mantysa)
0.5000000000000001	0 01...10 0...001
0.5000000000000002	0 01...10 0...010
0.5000000000000003	0 01...10 0...011
⋮	⋮
0.9999999999999997	0 01...10 1...101
0.9999999999999998	0 01...10 1...110
0.9999999999999999	0 01...10 1...111

Table 9: Rozkład liczb podwójnej precyzji w przedziale $[\frac{1}{2}, 1]$

Na podstawie uzyskanych wyników powtórzonego eksperymentu dla przedziału $[\frac{1}{2}, 1]$, możemy stwierdzić, że nasze założenie było poprawne, a liczby w zadanym przedziale są rozmieszczone równolegle z krokiem $\delta = 2^{-53}$.

Przedział $[2, 4]$

Przeprowadziwszy rozumowanie analogiczne do tego dla przedziału $[\frac{1}{2}, 1]$, możemy przyjąć, że dla przedziału $[2, 4]$ powinniśmy sprawdzić rozmieszczenie liczb z krokiem $\delta = 2^{-51}$.

Wartość	Postać binarna (znak bitu, cecha, mantysa)
2.0000000000000004	0 10...0 0...001
2.000000000000001	0 10...0 0...010
2.0000000000000013	0 10...0 0...011
⋮	⋮
3.9999999999999996	0 10...0 1...101
3.999999999999999	0 10...0 1...110
3.9999999999999987	0 10...0 1...111

Table 10: Rozkład liczb podwójnej precyzji w przedziale $[2, 4]$

Tutaj również możemy zauważyć, że liczby z zadanego przedziału są równolegle rozmieszczone dla założonej δ, która jest tutaj równa 2^{-51}.

Ogólne spostrzerzenie

Możemy zauważyć, że w arytmetyce zmiennopozycyjnej liczby w przedziałach $[2^i, 2^{i+1}]$ są równomiernie rozmieszczone z krokiem $\delta = 2^{-t+i}$, gdzie $t \equiv$ liczba bitów mantysy.
Z tego wynika fakt, że liczby w przedziałach $[2^i, 2^{i+1}]$ oraz $[2^j, 2^{j+1}]$ mają inne rozmieszczenie dla $i \neq j$. Dokładniej - im wieksza jest wartość i, tym "rzadziej" rozmieszczone są liczby w przedziale $[2^i, 2^{i+1}]$.

Zadanie 4 - Liczby odwrotne

Problem

a. Znaleźć w języku `Julia` liczbę zmiennopozycyjną x w precyzji `Float64` taką, że:

$$\begin{cases} x \epsilon (1,2) \\ x \cdot \frac{1}{x} \neq 1 \iff fl(x \cdot fl(\frac{1}{x})) \neq 1 \end{cases} \tag{2}$$

b. Znaleźć najmniejszą liczbę x spełniającą (2)

Rozwiązanie

Zaczynając od $x =$ `nextfloat(1)` (ponieważ $x > 1$) sprawdzać kolejne liczby zmiennoprzecinkowe (przypisując $x \leftarrow$ `nextfloat(x)`), dopóki $x < 2 \wedge x \cdot \frac{1}{x} = 1$.

Taki algorytm znajdzie najmniejszą liczbę x spełniającą warunki (2).

Program z rozwiązaniem: `ex4.jl`

Wyniki i obserwacje

Najmniejszą liczbą spełniającą warunki zadania jest $x = 1.000000057228997$.

Możemy zauważyć, że dokładność obliczeń (precyzja arytmetyki) ma znaczący wpływ na uzyskiwane wyniki, dlatego należy mieć to na uwadze, implementując algorytmy, by uniknąć lub zminimalizować wystąpienia takich błędów i ich kumulowania się

Zadanie 5 - Iloczyn skalarny

Problem

Zaimplementować w języku `Julia` algorytm obliczania iloczynu skalarnego dwóch wektorów, sumując przemnożone elementy na cztery sposoby: w przód, w tył, od największego do najmniejszego oraz od najmniejszego do najwięszego Algorytmy należy przetestować dla precyzji `Float32` oraz `Float32`.

Rozwiązanie

Program z rozwiązaniem: `ex5.jl`

Wyniki i obserwacje

Znając rzeczywistą wartość iloczynu skalarnego zadanych wektorów x oraz y:

$$S_{true} = 1.0065710699999998e - 11$$

Możemy obliczyć błąd względny uzyskanych wyników wg wzoru: $\delta = \frac{|S - S_{true}|}{S_{true}}$

Algorytm	S	δ
W przód	-0.4999443	$4.966805766328285e10$
W tył	-0.4543457	$4.5137965582009605e10$
Od największych	-0.5	$4.967359135506108e10$
Od najmniejszych	-0.5	$4.967359135506108e10$

Table 11: Wartości iloczynu skalarnego wraz z błędem względnym w precyzji `Float32`

Algorytm	S	δ
W przód	$1.0251881368296672e - 10$	9.184955313981629
W tył	$-1.5643308870494366e - 10$	16.54118664516592
Od największych	0.0	1.0
Od najmniejszych	0.0	1.0

Table 12: Wartości iloczynu skalarnego wraz z błędem względnym w precyzji `Float64`

Na podsawie wyników z *tabeli 12* możemy stwerdzić, że kolejność sumowania elementów w arytmetyce zmiennopozycyjnej może mieć duże znaczenie dla otrzymanego wyniku.
Dodatkowo zauważmy, że wektory x oraz y są do siebie prawie prostopadłe - ich iloczyn skalarny jest bliski 0. Zatem biorąc pod uwagę fakt, że otrzymane błędy sięgały nawet $\sim 1600\%$, możemy stwirdzić, że oblicznia bliskie 0 mogą generować duże błędy.

Zadanie 6 - Równoważne wunkcje

Problem

Wyznaczyć wartości zadanych funkcji:

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

W arytmetyce zmiennopozycyjnej w podwójnej precyzji dla $x = 8^i : i \, \epsilon \, \mathbb{N}^+$

Rozwiązanie

Program z rozwiązaniem: `ex6.jl`

Wyniki i obserwacje

Eksperyment przeprowadziłem dla $i \, \epsilon \, \{1, 2, ..., 16\}$.

Exponent: i	$f(x)$	$g(x)$
1	0.0077822185373186414	0.0077822185373187065
2	0.00012206286282867573	0.00012206286282875901
3	$1.9073468138230965e-6$	$1.907346813826566e-6$
4	$2.9802321943606103e-8$	$2.9802321943606116e-8$
5	$4.656612873077393e-10$	$4.6566128719931904e-10$
6	$7.275957614183426e-12$	$7.275957614156956e-12$
7	$1.1368683772161603e-13$	$1.1368683772160957e-13$
8	$1.7763568394002505e-15$	$1.7763568394002489e-15$
9	0.0	$2.7755575615628914e-17$
10	0.0	$4.336808689942018e-19$
11	0.0	$6.776263578034403e-21$
12	0.0	$1.0587911840678754e-22$
13	0.0	$1.6543612251060553e-24$
14	0.0	$2.5849394142282115e-26$
15	0.0	$4.0389678347315804e-28$
16	0.0	$6.310887241768095e-30$

Table 13: Wartości równoważnych funkcji f oraz g

Na podstawie powyższej tabeli możemy stwierdzić, że mimo matematycznej równoważności funkcji f i g, nie są one równoznaczne w arytmetyce zmiennopozycyjnej.
W tym przypadku bardziej wiarygodne wyniki zwraca funkcja g. Niedokładność funkcji f może wynikać z tego, że wykonuje ona operację odejmowania bliskich siebie wartości.

Zadanie 7 - Przybliżenie pochodnej

Problem

Wyznaczyć w języku `Julia` przybliżone wartości pochodnej funkcji $f(x) = \sin x + \cos 3x$, korzystając z wzoru:

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

Dla $x_0 = 1 \wedge h = 2^{-n} : n \, \epsilon \, \{0, 1, ..., 54\}$

Rozwiązanie

Program z rozwiązaniem: `ex7.jl`

Wyniki i obserwacje

Wiedząc, że $f'(x0) = \cos(x_0) - 3 \cdot sin(3x_0)$ możemy porównać otrzymane przybliżenia pochodnej funkcji f z jej rzeczywistą wartością oraz obliczyć błąd bezwzględny tych przybliżeń.

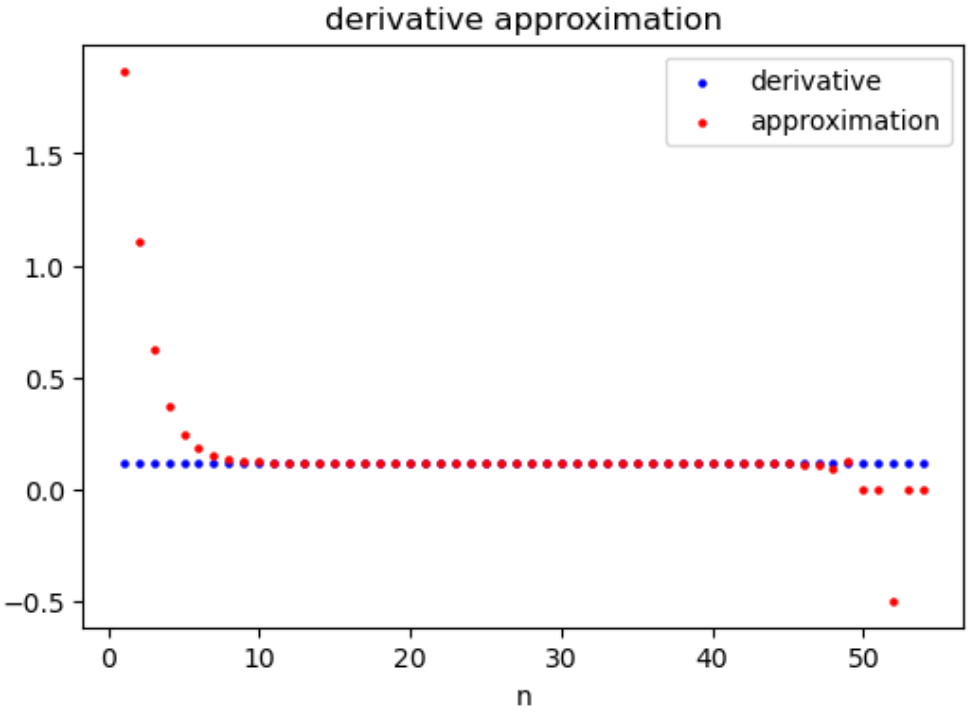

Figure 1: Przybliżenie pochodniej funkcji f

Na podstawie *wykresu 1* mozemy zauważyć, że na początku zmniejszanie wartości h powoduje bardzo szybki wzrost dokładności przybliżenia, jednak od pewnego momentu przybliżenie to przestaje być bardziej dokładne, a nawet zaczyna być coraz mniej, co wynika z tego, że $h \approx 0 \Rightarrow fl(1+h) = 1$.
Można to zauważyć także na *wykresie 2*, który bardzo dobrze pokazuje spadek dokładności przybliżenie pochodnej funkcji f dla bardzo małych wartości h.

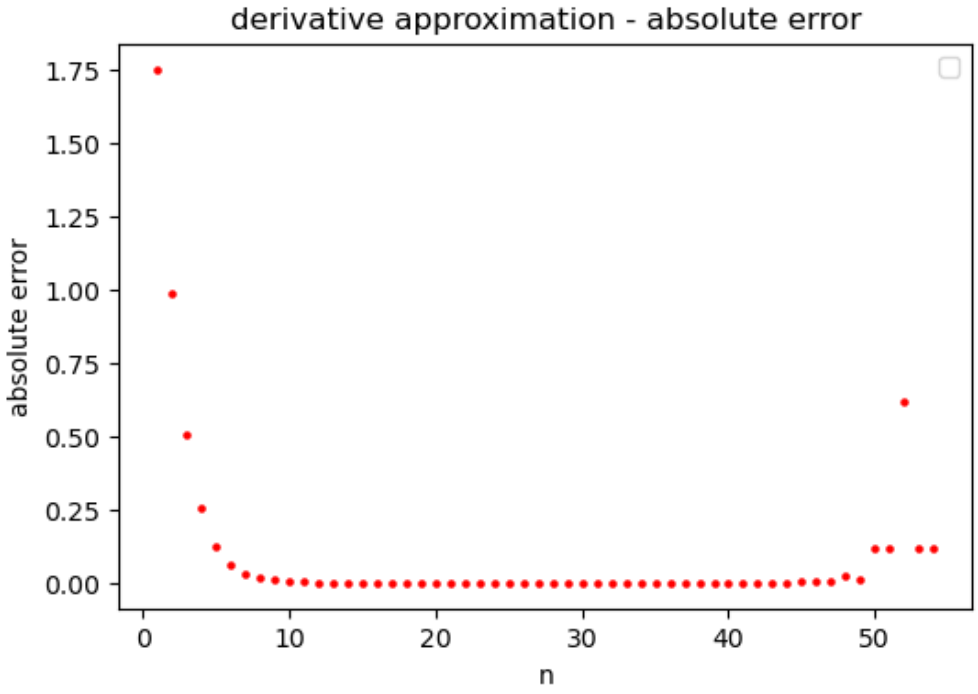

Figure 2: Błąd bezwzględny przybliżenia pochodnej funkcji f