Akademia Górniczo-Hutnicza
Wydział Informatyki
Kierunek Informatyka

Metody Obliczeniowe w Nauce i Technice

Laboratorium 1

Analiza błędów

Wojciech Michaluk, Kyrylo Iakymenko

Kraków, 1 marca 2024

1 Wprowadzenie

Podczas tego laboratorium zrealizujemy 2 zadania, na podstawie których będziemy przeprowadzać analizę błędów - jest ich bowiem wiele rodzajów.

W zadaniu pierwszym skupimy się na błędach obliczeniowych, numerycznych i błędach metody. Przedstawimy je na przykładzie obliczania pochodnej wybranej funkcji w punkcie, przybliżając ją za pomocą ilorazu różnicowego i wykonamy obliczenia dla różnych wartości kroku. Skorzystamy z odpowiednich zależności, aby obliczyć błędy i zaprezentujemy je na wspólnym wykresie.

Z kolei zadanie drugie polega na obliczania kolejnych wyrazów ciągu rekurencyjnego zdefiniowanego równaniem różnicowym, przyjmując różne reprezentacje liczb zmiennoprzecinkowych. Dla każdej z nich przeanalizujemy otrzymane wartości wyrazów ciągu i porównamy z teoretycznymi, następnie przedstawimy odpowiednio błędy względne w każdej z reprezentacji.

2 Zadanie 1

2.1 Opis zadania

W zadaniu należy obliczyć wartość pochodnej funkcji, używając najpierw wzoru na różnicę prawostronną:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{1}$$

a następnie używając wzoru różnic centralnych, czyli

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{2}$$

Rozważaną funkcją jest $tan(x)$ w punkcie $x = 1$, natomiast przyjmowane wartości h wynoszą $10^{-k}, k = 0, 1, ..., 16$.

Kolejnym etapem jest wyznaczenie błędu metody, błędu numerycznego i błędu obliczenowego w zależności od h oraz odpowiednie ich przedstawienie na wspólnym wykresie i wyciągnięcie wniosków.

Do obliczenia wspomnianych błędów pomocne będzie skorzystanie z tożsamości

$$tan'(x) = 1 + tan^2(x), \tag{3}$$

która pozwala wyznaczyć rzeczywistą wartość pochodnej.

2.2 Podejście pierwsze - różnica prawostronna

W tym przypadku zachodzi zależność [1]

$$E(h) \leq \frac{Mh}{2} + \frac{2\epsilon}{h}, \tag{4}$$

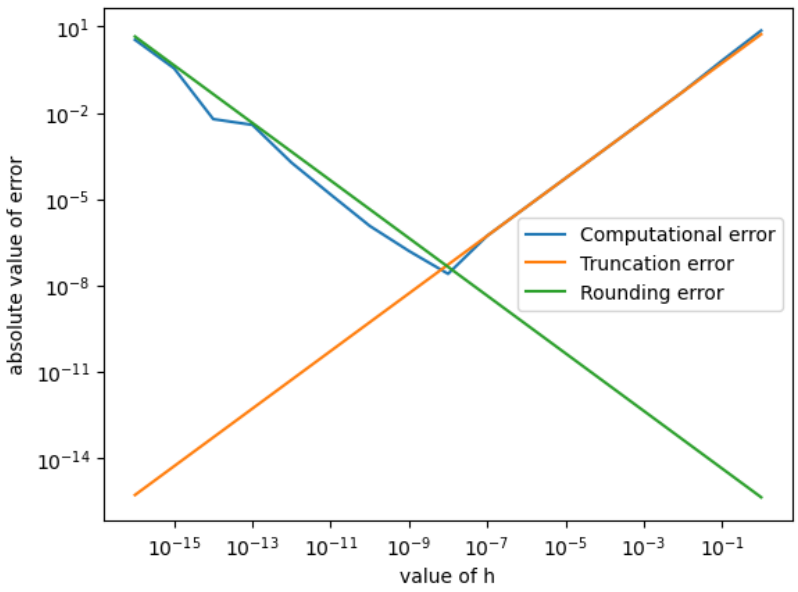
gdzie:

- błąd metody, zwany także błędem obcięcia (*truncation error*) jest wyrażony wzorem $\frac{Mh}{2}$, natomiast M to przybliżona wartość drugiej pochodnej rozważanej funkcji w punkcie $x = 1$. Tutaj $M \approx 10.669859$.
- błąd numeryczny (*rounding error*) to $\frac{2\epsilon}{h}$, przy czym ϵ oznacza precyzję przedstawienia liczby w przyjętej reprezentacji zmiennoprzecinkowej, tzw. ϵ_{mach} - najmniejsza liczba, dla której (jeszcze) jest spełniony warunek $1 + \epsilon > 1$, biorąc pod uwagę reprezentację zmiennoprzecinkową wyniku operacji $1 + \epsilon$. W tym zadaniu używamy reprezentacji float64, $\epsilon \approx 2.220446 \cdot 10^{-16}$.

[1] Wzór zaczerpnięty z [1].

- błąd obliczeniowy $E(h)$ (*computational error*) jest to różnica między uzyskanym wynikiem a wartością obliczoną ze wzoru (3).

Przedstawiamy wykresy wartości bezwzględnych błędów na wspólnym wykresie, przy czym na obu osiach użyto skali logarytmicznej.



Rysunek 1: Wykres wspólny wartości bezwzględnych błędów - różnica prawostronna

Zauważmy, że wykresy błędów zachowują się zgodnie z przewidywaniami opartymi na zależności (4). Ponadto można stwierdzić, że wykres błędu obliczeniowego ma w pewnym punkcie minimum - mianowicie w 10^{-8}, natomiast należy wziąć poprawkę na fakt, że punkty wykresu zostały połączone prostymi liniami, aby zwiększyć czytelność. Poniżej przedstawiamy obliczenia prowadzące do wyznaczenia dokładnej wartości h_{min}, dla której prawa strona nierówności (4) przyjmuje najmniejszą wartość. Mamy zatem

$$\frac{\partial(\frac{Mh}{2} + \frac{2\epsilon}{h})}{\partial h} = 0 \tag{5}$$

$$\frac{M}{2} - \frac{2\epsilon}{h^2} = 0 \tag{6}$$

$$h^2 = \frac{4\epsilon}{M} \tag{7}$$

$$h_{min} = h = 2\sqrt{\frac{\epsilon}{M}} \tag{8}$$

Podstawiając wartości ϵ i M, uzyskujemy $h_{min} \approx 9.123695 \cdot 10^{-9}$.

Przejdźmy zatem do obliczenia wartości bezwzględnej błędu względnego (obliczeniowego) w punkcie h_{min}. Wyliczamy ją ze wzoru $\left|\frac{E(h_{min})}{tan'(1)}\right|$ i otrzymujemy

$$\left|\frac{E(h_{min})}{tan'(1)}\right| \approx 5.33866349 \cdot 10^{-9}$$

.

2.3 Podejście drugie - różnica centralna

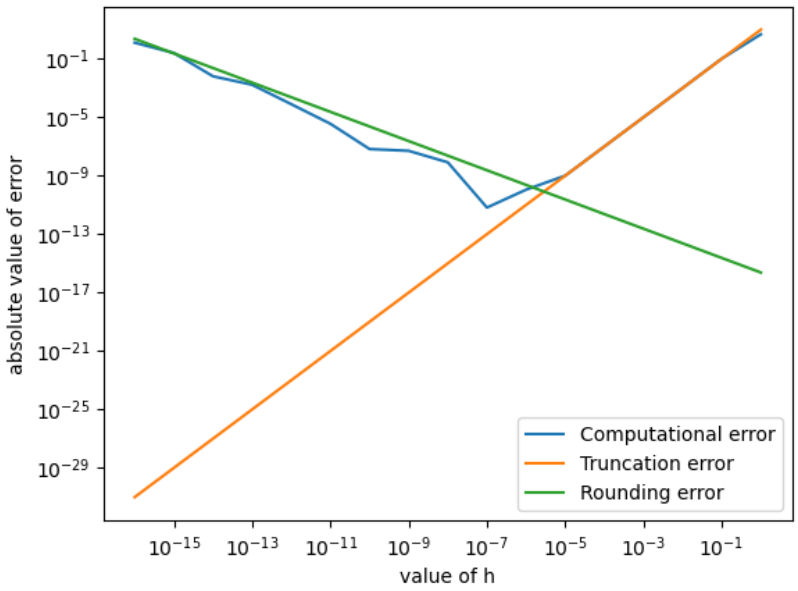
Przy tym podejściu zachodzi zależność [2]

$$E(h) \leq \frac{Mh^2}{6} + \frac{\epsilon}{h}, \tag{9}$$

gdzie:

- błąd metody jest wyrażony wzorem $\frac{Mh^2}{6}$ i w tym przypadku $M \approx 56.7029999$,
- błąd numeryczny to $\frac{\epsilon}{h}$ (ϵ ma taką samą wartość jak w pierwszym podejściu),
- błąd obliczeniowy $E(h)$ jest, tak jak poprzednio, różnicą między uzyskanym wynikiem a wartością obliczoną ze wzoru (3).

Przedstawiamy wykresy wartości bezwzględnych błędów na wspólnym wykresie, przy czym na obu osiach użyto skali logarytmicznej.



Rysunek 2: Wykres wspólny wartości bezwzględnych błędów - różnica centralna

Podobnie jak w pierwszym podejściu, tutaj także wykresy zachowują się zgodnie z przewidywaniami wedle zależności (9) oraz wykres błędu obliczeniowego ma w pewnym punkcie minimum, które tym razem graficznie przypada w 10^{-7}. Poniżej przedstawiamy obliczenia prowadzące do wyznaczenia dokładnej wartości h_{min}, dla której prawa strona nierówności (9) przyjmuje najmniejszą wartość. Mamy zatem

$$\frac{\partial(\frac{Mh^2}{6} + \frac{\epsilon}{h})}{\partial h} = 0 \tag{10}$$

$$\frac{Mh}{3} - \frac{\epsilon}{h^2} = 0 \tag{11}$$

$$h^3 = \frac{3\epsilon}{M} \tag{12}$$

$$h_{min} = h = \sqrt[3]{\frac{3\epsilon}{M}} \tag{13}$$

Podstawiając wartości ϵ i M, uzyskujemy $h_{min} \approx 2.273274 \cdot 10^{-6}$.

Możemy teraz obliczyć wartość bezwzględną błędu względnego w wyznaczonym punkcie h_{min}. Postępując analogicznie, w tym przypadku wynosi ona $\approx 2.53340126 \cdot 10^{-11}$.

[2] Wzór zaczerpnięty z [1].

3 Zadanie 2

3.1 Opis zadania

Celem tego ćwiczenia jest zrozumienie sposobu rozwiązywania równań różnicowych przy użyciu iteracji oraz analiza wpływu precyzji obliczeń na wyniki. Rozważmy następujące równanie różnicowe:

$$x_{k+1} = 2.25x_k - 0.5x_{k-1},$$
$$\text{gdzie } x_0 = \frac{1}{3}, x_1 = \frac{1}{12}.$$

Zadanie polega na wygenerowaniu pierwszych n wyrazów tego ciągu oraz analizie zachowania się ciągu w zależności od precyzji użytych obliczeń.

Wykonamy obliczenia dla 3 przypadków:

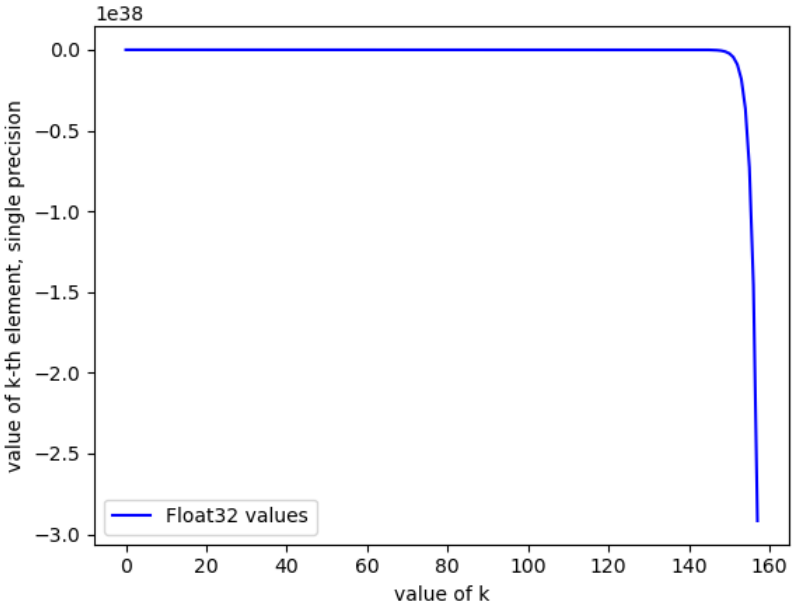
1. używając pojedynczej precyzji (float32) oraz przyjmując $n = 225$,
2. używając podwójnej precyzji (float64) oraz przyjmując $n = 60$,
3. używając reprezentacji *Fraction* z biblioteki *fractions* oraz przyjmując $n = 225$.

Dalej porównamy wyniki uzyskane przy użyciu wymienionych reprezentacji liczb zmiennoprzecinkowych z wartościami obliczonymi przy użyciu postaci jawnej ciągu:

$$x_k = \frac{4^{-k}}{3}.$$

3.2 Uzyskane wartości wyrazów ciągu

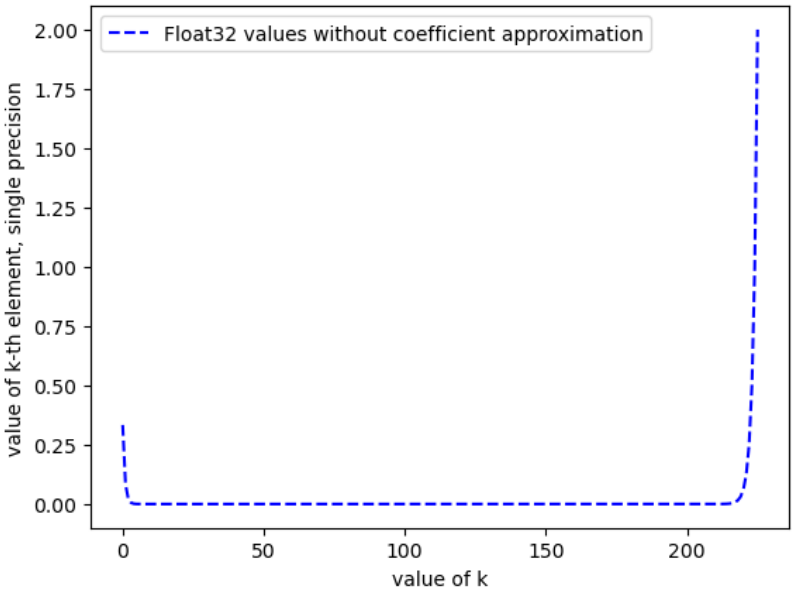
Najpierw opiszemy wyniki uzyskane dla float32 i $n = 225$. Widzimy, że po pewnym



Rysunek 3: Nieskalowany wykres wartości ciągu dla float32

wyrazie ciągu (dokładniej po 9.), wszystkie wartości schodzą poniżej 0 i ciąg jest stale malejący. Wyjaśnić to można poprzez zjawisko zwane *underflow* [2]. W pewnym momencie brakuje dokładności reprezentacji float32, żeby wystarczająco dokładnie zapisać wynik danego obliczenia i wartość jest zaokrąglona do najbliższej możliwej - w tym przypadku okazuje się, że w pewnym momencie jest nią liczba mniejsza od 0 w reprezentacji float32. **Uwaga!** Dla $k > 157$ wartości są tak małe, że wynik obliczenia jest nieokreślony ($-\infty$, dla $k > 159$ jest to nan). Zatem na powyższym wykresie nie są one obecne.

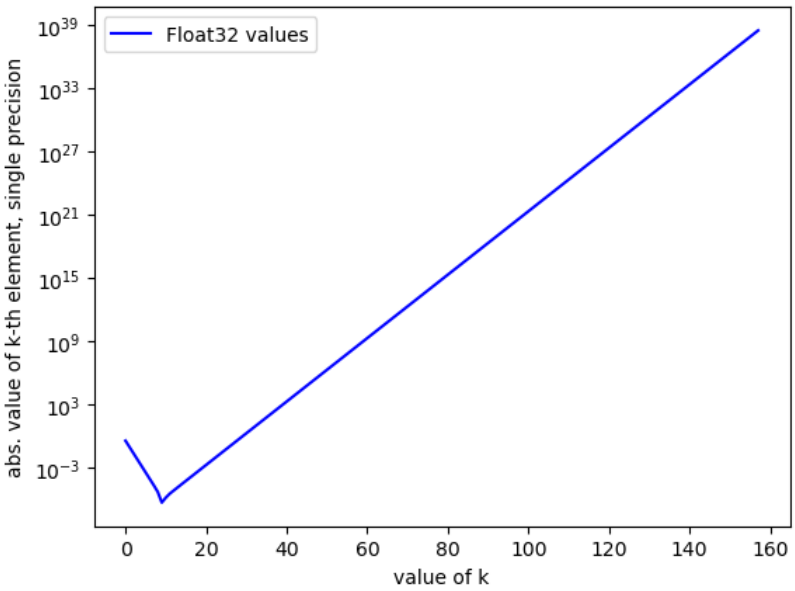
Dla porównania przedstawiamy wykres, gdy nie używamy funkcji *np.float32* na współczynnikach ciągu i dostajemy bardziej sensowny, przypominający float64 wykres.



Rysunek 4: Nieskalowany wykres wartości ciągu dla float32 bez przybliżenia współczynników

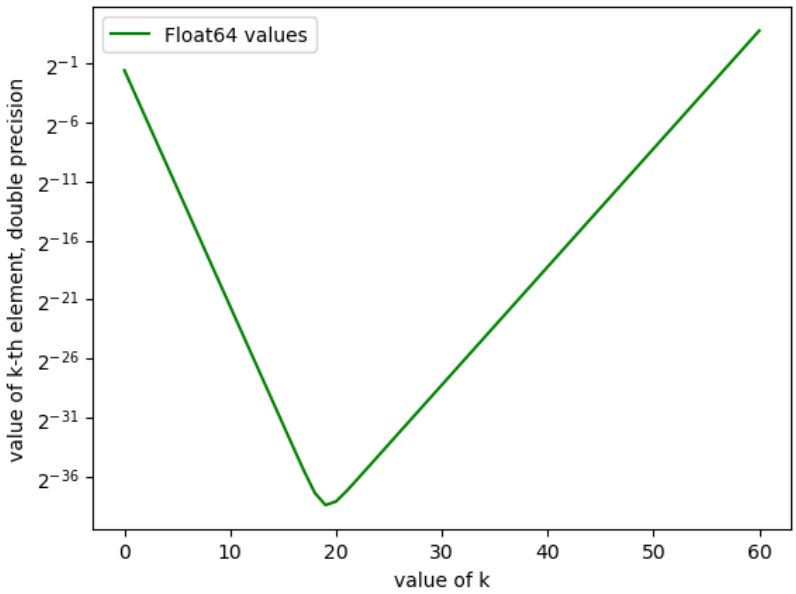
Moglibyśmy wykorzystać ten sposób do obliczenia wyrazów ciągu, ale wtedy okazuje się, że float32 jest bardziej dokładny od float64 (w dalszych testach błędu względnego float32 miałoby mniejszy błąd). Uznajemy to za niezbyt sensowne i postanowiliśmy testować wszystkie funkcje, wykorzystując przybliżenie współczynników ciągu - nie patrzymy na to, że float32 ucieka do $-\infty$, a float64 do $+\infty$.

Poniżej przedstawiony jest wykres logarytmiczny wartości bezwzględnej wyrazów ciągu w reprezentacji float32 (wykorzystujemy wartość bezwzględną dla wygody porównania wykresów z ciągiem float64 w przyszłości).



Rysunek 5: Wykres symlog wartości ciągu dla float32

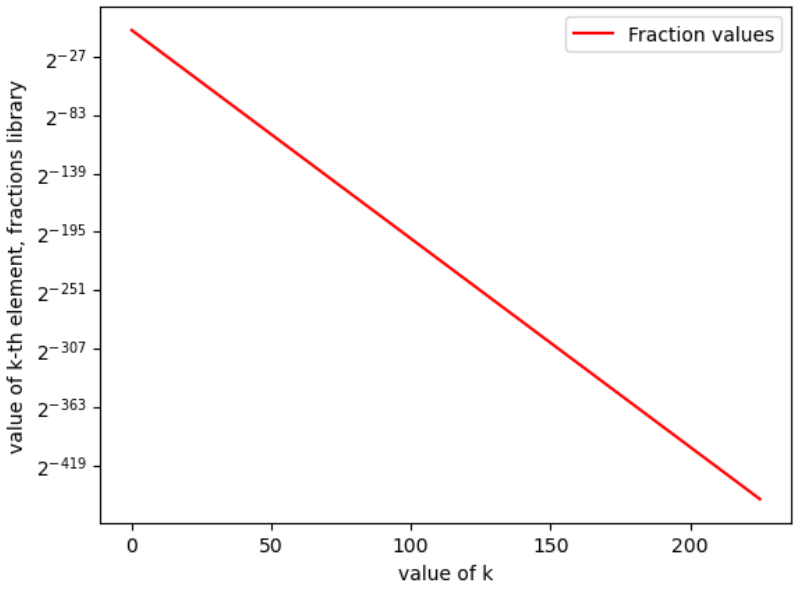
Poniżej prezentujemy logarytmiczny wykres wartości ciągu dla reprezentacji float64.



Rysunek 6: Logarytmiczny wykres wartości ciągu dla float64

Jak widać, podobnie do float32 (w drugim rozważanym przypadku), float64 w pewnym momencie osiąga minimum i zaczyna rosnąć. Znowu związane to jest z tym, że w pewnym momencie następuje underflow, brakuje precyzji reprezentacji float64, żeby dokładnie przedstawić liczbę i wykorzystywane są przybliżenia. Z czasem coraz większa ilość przybliżeń się sumuje i dostajemy wyrazy coraz bardziej różniące się od wartości rzeczywistej. W pewnym momencie błąd reprezentacji powoduje, że wartość x_k jest większa od x_{k-1} i wtedy ciąg rośnie w nieskończoność.

Teraz spójrzmy na wykres dla reprezentacji z biblioteki fractions.

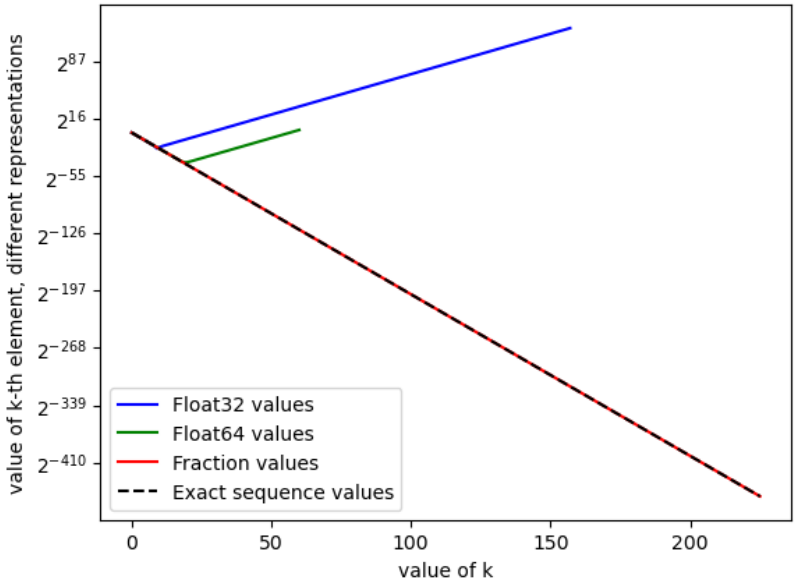


Rysunek 7: Logarytmiczny wykres wartości ciągu dla fractions

Widać, że to jest najbardziej dokładna prezentacja. Ciąg wartości jest malejący i większy od zera oraz reprezentuje dokładnie wartości wyliczone z postaci jawnej ciągu (zobaczymy to poniżej, przy opracowaniu błędów względnych). Prawdopodobnie związane to jest z tym, że postać jawna ciągu to $x_k = \frac{4^{-k}}{3}$, czyli każdy wyraz ciągu da się przedstawić dokładnie w postaci ułamka i patrząc na to, co Fractions robi pod spodem (mnoży, ewentualnie dodaje mianowniki i liczniki, a potem skraca ułamek), to taką dokładność można

łatwo uzasadnić tym, że w każdym momencie działamy na liczbach całkowitych (mianowniku i liczniku). Dla naszego przypadku, kiedy wyrazy ciągu są wymierne i w dodatku do tego łatwo znaleźć k-ty wyraz w postaci ułamka, Fraction jest najbardziej sensowną reprezentacją.

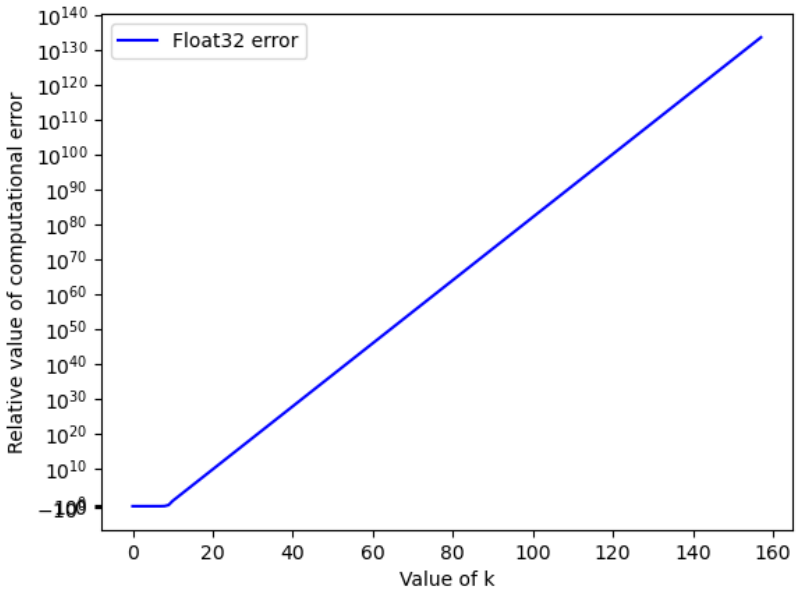
Poniżej przedstawiamy porównanie wszystkich wyników. Zauważamy między innymi, że rzeczywiste wartości ciągu w zasadzie nakładają się na wykres wartości uzyskanych z Fraction.



Rysunek 8: Wykres wspólny wartości ciągu dla różnych precyzji

3.3 Błędy względne

Przechodząc teraz do opracowania błędów względnych naszych reprezentacji, zaczniemy znów od float32.

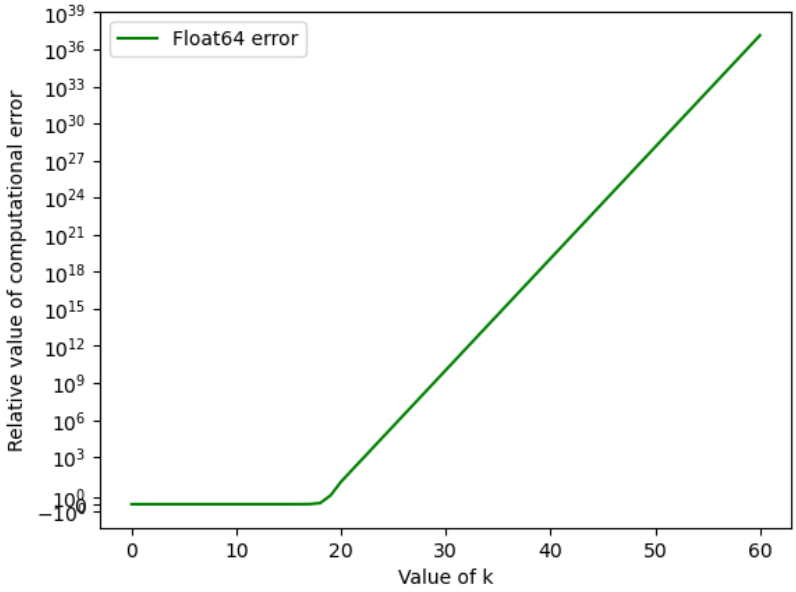


Rysunek 9: Logarytmiczny wykres błędu względnego wyrazów ciągu dla float32

Tak jak przewidywaliśmy przy wyjaśnieniu zachowania wykresów float32 i float64, do pewnego momentu obliczenia są zapisywane w miarę dokładnie, ale po jakimś czasie ograniczona precyzja powoduje underflow i błąd zaczyna rosnąć eksponencjalnie (wykres

jest w skali logarytmicznej). Można zatem stwierdzić, że reprezentacja float32 dla naszego ciągu była dokładna tylko dla pierwszych 10 wyrazów.

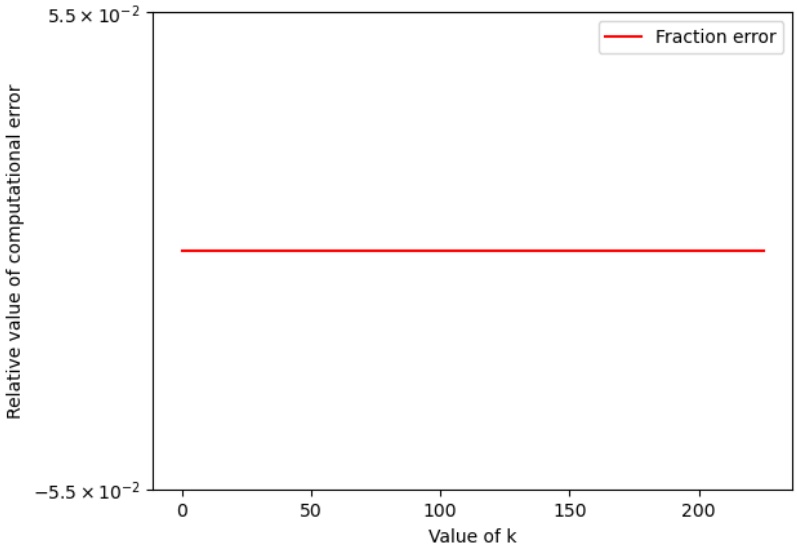
Podobną analizę przeprowadzamy dla reprezentacji float64.



Rysunek 10: Logarytmiczny wykres błędu względnego wyrazów ciągu dla float64

Widzimy podobną do float32 sytuację - do pewnego momentu nasze obliczenia mają dużą precyzję, ale po tym, jak dochodzimy do momentu, gdy wiele underflow z rzędu powoduje zmniejszenie precyzji, błąd rośnie bardzo szybko.

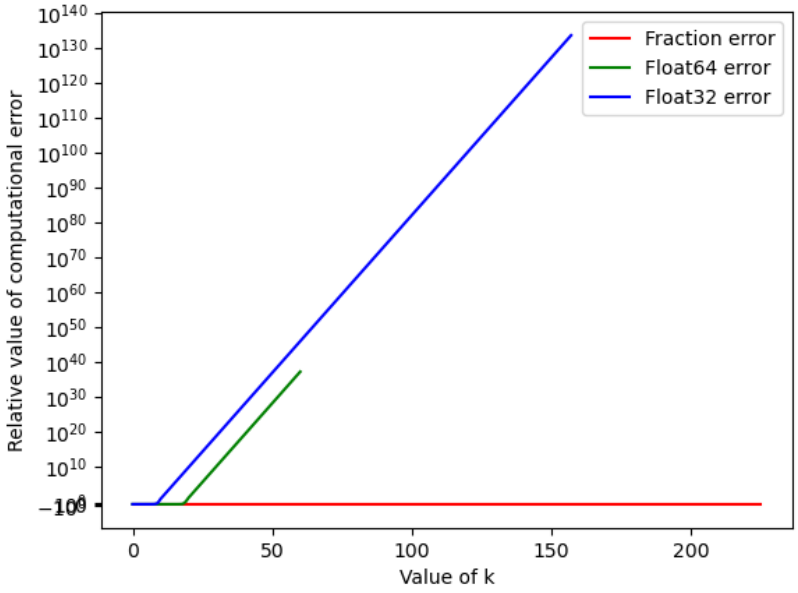
Fractions wyróżnia się spośród badanych reprezentacji swoim zerowym błędem.



Rysunek 11: Logarytmiczny wykres błędu względnego wyrazów ciągu dla fractions

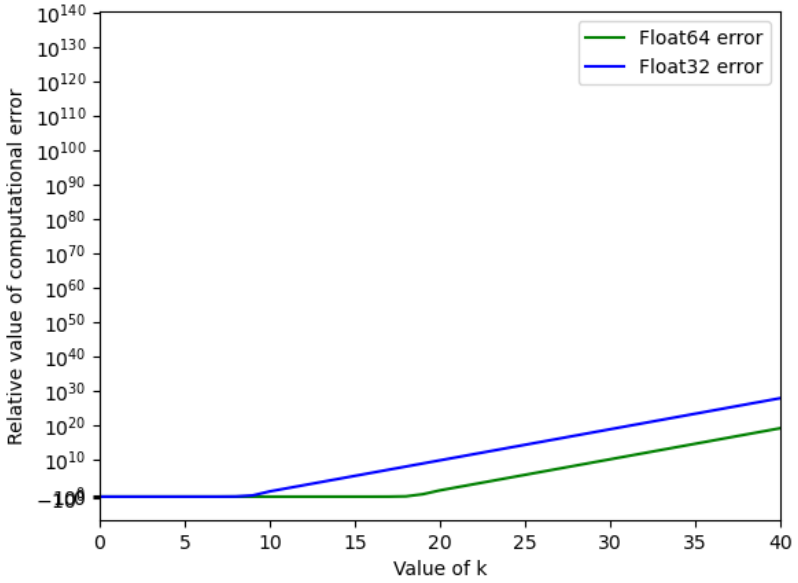
Ponownie nasze spostrzeżenia odnośnie dobrej precyzji fractions w tym zadaniu (między innymi ze względu na to, że wyraz ogólny ciągu da się w prosty sposób opisać za pomocą odpowiedniego ułamka) się sprawdzają.

Podsumujmy nasze rozważania, przedstawiając wspólny wykres wszystkich błędów względnych.



Rysunek 12: Logarytmiczny wykres błędu względnego wyrazów ciągu dla wszystkich precyzji

Na tym wykresie widzimy, na ile floaty zachowują się w podobny sposób. Poniżej na kolejnym wykresie lepiej widać moment, gdy każdy z floatów zaczyna szybko tracić precyzję.



Rysunek 13: Porównanie wartości błędu względnego dla float32 i float64

Z opracowania danych wynika, że float32 osiąga swoje minimum dla 9. wyrazu, a float64 dla 19. wyrazu. To jest mniej więcej zgodne z przedstawieniem tego, jaką precyzję ma każda z tych reprezentacji, gdyż float64 używa dwa razy więcej bitów i wartość minimalna jest rzędu 10^{-12}, a w przypadku float32 jest to tylko 10^{-7} (dla porównania, najmniejszy wyraz ciągu obliczonego za pomocą fractions to około 10^{-136}). Ciekawym i może trochę kontrintuicyjnym spostrzeżeniem jest to, że zarówno błędy względne float32, jak i float64, mają podobne tempo wzrostu po momencie, kiedy zaczynają gwałtownie rosnąć.

4 Podsumowanie

W ramach tego laboratorium przeprowadziliśmy analizę numerycznych metod obliczania pochodnej funkcji i wyrazów ciągu rekurencyjnego oraz dokonaliśmy analizy błędów związanych z tymi metodami.

W pierwszym zadaniu zaimplementowaliśmy numeryczną metodę obliczania pochodnej funkcji, używając przybliżenia za pomocą ilorazu różnicowego. Przetestowaliśmy tę metodę dla funkcji tangens oraz wyznaczyliśmy błąd, porównując wyniki numeryczne z prawdziwą wartością pochodnej. Następnie przedstawiliśmy wykresy wartości bezwzględnej błędu metody, błędu numerycznego oraz błędu obliczeniowego w zależności od kroku h. Zauważyliśmy, że błąd obliczeniowy osiąga minimum dla określonej wartości h, co zgadza się z teoretycznymi przewidywaniami, jakie wynikają z analizy błędów numerycznych.

W drugim zadaniu zaimplementowaliśmy program generujący wyrazy ciągu zdefiniowanego równaniem różnicowym. Przeprowadziliśmy obliczenia dla różnych precyzji (pojedynczej, podwójnej oraz reprezentacji ułamków) i narysowaliśmy wykresy wartości ciągu w zależności od k. Dodatkowo, przedstawiliśmy wykres wartości bezwględnej błędu względnego w zależności od k. Zaobserwowaliśmy, że wartości ciągu stale maleją wraz ze wzrostem k jedynie dla reprezentacji fractions, a float32 i float64 zachowują precyzję tylko dla ograniczonej ilości początkowych wyrazów.

Literatura

[1] Materiały pomocnicze do laboratorium zamieszczone na platformie Teams w katalogu *lab01/lab1-intro.pdf*.

[2] Artykuł o ‘Arithmetic underflow‘ z wikipedii `https://en.wikipedia.org/wiki/Arithmetic_underflow`.