Marek Małek, Marcin Serafin, 05.03.2024

Laboratorium 01

Analiza Błędów

1. Treść zadań

Zadanie 1. Oblicz przybliżoną wartość pochodnej funkcji, używając wzoru

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

Sprawdź działanie programu dla funkcji $\tan(x)$ oraz $x = 1$. Wyznacz błąd, porównując otrzymaną wartość numerycznej pochodnej z prawdziwą wartością. Pomocna będzie tożsamość $\tan'(x) = 1 + \tan^2(x)$.
Na wspólnym rysunku przedstaw wykresy wartości bezwzględnej błędu metody, błędu numerycznego oraz błędu obliczeniowego w zależności od h dla $h = 10^{-k}$, $k = 0, \ldots, 16$. Użyj skali logarytmicznej na obu osiach. Czy wykres wartości bezwzględnej błędu obliczeniowego posiada minimum?
Porównaj wyznaczoną wartość $h_{\min}$ z wartością otrzymaną ze wzoru

$$h_{\min} \approx 2\sqrt{\epsilon_{\text{mach}}/M}, \text{ gdzie } M \approx |f''(x)| \quad . \tag{2}$$

Powtórz ćwiczenie używając wzoru różnic centralnych

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad . \tag{3}$$

Porównaj wyznaczoną wartość $h_{\min}$ z wartością otrzymaną ze wzoru

$$h_{\min} \approx \sqrt[3]{3\epsilon_{\text{mach}}/M}, \text{ gdzie } M \approx |f'''(x)| \quad . \tag{4}$$

Zadanie 2. Napisz program generujący pierwsze n wyrazów ciągu zdefiniowanego równaniem różnicowym:

$$x_{k+1} = 2.25x_k - 0.5x_{k-1}$$

z wyrazami początkowymi:

$$x_0 = \frac{1}{3} \qquad x_1 = \frac{1}{12} \ .$$

Wykonaj obliczenia

- używając pojedynczej precyzji oraz przyjmując $n = 225$
- używając podwójnej precyzji oraz przyjmując $n = 60$
- używając reprezentacji z biblioteki `fractions` oraz przyjmując $n = 225$.

Narysuj wykres wartości ciągu w zależności od k. Użyj skali logarytmicznej na osi y (pomocna będzie funkcja `semilogy`). Następnie narysuj wykres przedstawiający wartość bezwzględną błędu względnego w zależności od k.
Dokładne rozwiązanie równania różnicowego:

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

maleje wraz ze wzrostem k. Czy otrzymany wykres zachowuje się w ten sposób? Wyjaśnij otrzymane wyniki.

Unikaj pętli `for` na rzecz kodu wektorowego oraz funkcji uniwersalnych (np. `np.tan` zamiast `math.tan`).

2. Argumentacja rozwiązania zadań, fragmenty algorytmu, kodu ilustrującego rozwiązanie

Zadania rozwiązano w języku Python. Użyte biblioteki: **numpy**, **matplotlib**, **pandas**.

2.1 Rozwiązanie zadania 1

- zdefiniowano stałe: epsilon oraz wartości parametru h

```
epsilon = np.finfo(float).eps
h_values = np.power(10, -np.arange(17), dtype=float)
```

- zaimplementowano funkcje pomocnicze

```
# testowana funkcja
def f(x):
  return np.tan(x)

# tożsamość pochodnej funkcji tan(x)
def f_prime(x):
  return 1 + np.tan(x)**2

# druga pochodna funkcji tan(x)
def f_prime_prime(x):
  return 2*np.tan(x)/(np.cos(x)**2)

# trzecia pochodna funkcji tan(x)
def f_prime_prime_prime(x):
  return 2*(1/np.cos(x)**2)**2 + 2*np.tan(x)*(-2*np.tan(x)/np.cos(x)**2)

# metoda różnic skończonych
def forward_diff_method(x, h, f):
  return (f(x + h) - f(x)) / h

# metoda różnic centralnych
def central_diff_method(x, h, f):
  return (f(x+ h)-f(x-h))/(2*h)

# wzór nr (2)
def forward_h_min():
  return 2*np.sqrt(epsilon/abs(f_prime_prime(1)))

# wzór nr (4)
def central_h_min():
  return np.cbrt(3*epsilon/abs(f_prime_prime_prime(1)))
```

- zaimplementowano funkcję zwracającą wartości błędu metody, błędu obliczeniowego oraz błędu numerycznego dla wcześniej zadanych wartości parametru h w zależności od wybranej metody. Funkcja także zwraca minimum z wartości bezwzględnej błędu obliczeniowego

```
def get_errors(f, M, method):

  values = [method(1, h, f) for h in h_values]
  h_min = min(values)
  computational_errors = abs(values - f_prime(1))
  if (method == forward_diff_method):
      rounding_errors = [2*epsilon/h for h in h_values]
      truncation_errors = [M*h/6 for h in h_values]
  else:
      rounding_errors = [epsilon/h for h in h_values]
      truncation_errors = [(M*h**2)/6 for h in h_values]

  return computational_errors, rounding_errors, truncation_errors, h_min
```

- napisano wizualizację rozwiązania dla obu metod
    - metoda różnic skończonych

```
forward_computational_error,
forward_rounding_error,
forward_trucation_error,
forward_experimental_h_min = get_errors(
 f,
 abs(f_prime_prime(1)),
 forward_diff_method
)

plt.figure(figsize=(16, 6))
plt.loglog(h_values, forward_computational_error,
           label='Approximation Error', marker='.')
plt.loglog(h_values, forward_rounding_error,
           label='Rounding Erorr', marker='.')
plt.loglog(h_values, forward_trucation_error,
           label='Truncation Error', marker='.')
plt.xlabel('h')
plt.ylabel('Absolute Error')
plt.title('Forward Difference Method')
plt.legend()
plt.show()
```

- metoda różnic centralnych

```
central_computational_error,
central_rounding_error,
central_trucation_error,
central_experimental_h_min = get_errors(
 f,
 abs(f_prime_prime(1)),
 central_diff_method
)

plt.figure(figsize=(16, 6))
plt.loglog(h_values, central_computational_error,
           label='Approximation Error', marker='.')
plt.loglog(h_values, central_rounding_error,
           label='Rounding Erorr', marker='.')
plt.loglog(h_values, central_trucation_error,
           label='Truncation Error', marker='.')
plt.xlabel('h')
plt.ylabel('Absolute Error')
plt.title('Central Difference Method')
plt.legend()
plt.show()
```

- napisano zestawienie wartości h_{min} z minimum wartości błędu obliczeniowego ze względu na metodę

```
df = pd.DataFrame({
  "eperimental values": [forward_experimental_h_min,
central_experimental_h_min],
  "expected values": [forward_h_min(), central_h_min()]
 })\
  .style.relabel_index(["forward method", "central method"], axis=0)\
  .format(precision=30)
```

2.2 Rozwiązanie zadania 2

- podobnie jak w przypadku zadania 1 zdefiniowano stałe określające długości ciągu dla poszczególnych precyzji.

```
# dla pojedyńczej precyzji
n_single = 225
# dla podwójnej precyzji
n_double = 60
# dla reprezentacji z biblioteki Fractions
n_fraction = 225
```

- zdefiniowano funkcje pomocnicze

```
# funkcja wyliczająca i-ty wyraz ciągu o zadanej precyzji precision w
oparciu o wyrazy ciągu x_k_values
def x_k(i, x_k_values, precision):
   if (i == 0):
       return x_k_values[0]
   elif (i == 1):
       return x_k_values[1]
   else:
       return precision(2.25)*x_k_values[i-1]-precision(0.5)*x_k_values[i-2]

# funkcja generująca n początkowych wyrazów ciągu o zadanej precyzji
precision
def generate_n_elements(n, precision):
    x_k_values = np.zeros(n, dtype=precision)
    x_k_values[0] = precision(1/3)
    x_k_values[1] = precision(1/12)
    for i in range(2, n):
        x_k_values[i] = (x_k(i, x_k_values, precision))
    return x_k_values

# funkcja wyliczająca n dokładnych rozwiązań równania różnicowego
def expected_values(n):
    expected_x_k_values = np.zeros(n)
    for i in range(n):
        expected_x_k_values[i] = (4**(-i))/3
    return expected_x_k_values
```

- wykonano obliczenia wartości ciągu dla zadanych precyzji i długości, obliczono wartości bezwględne błędu względnego

```
x_single = generate_n_elements(n_single, np.float32)
x_double = generate_n_elements(n_double, np.float64)
x_fractions = generate_n_elements(n_fraction, Frac)
x_expected = expected_values(n_single)

error_single = np.abs((x_single - x_expected) / x_expected)
error_double = np.abs((x_double - x_expected[:n_double])  /
x_expected[:n_double])
error_fractions = np.abs((x_fractions - x_expected) / x_expected)
```

- napisano wizualizację rozwiązania
    - wykres wartości bezwględnej błędu względnego

```
plt.semilogy(np.arange(1, n_single+1), x_single, label='Single
Precision')
plt.semilogy(np.arange(1, n_double+1), x_double, label='Double
Precision')
plt.semilogy(np.arange(1, n_fraction+1), x_fractions,
label='Fractions')
plt.semilogy(np.arange(1, n_fraction+1), x_expected, label='Expected
Values')
plt.title('Sequence Values')
plt.xlabel('k')
plt.ylabel('Sequence Value')
plt.legend()
plt.grid(True)
plt.show()
```

    - wykres rozwiązań równania róznicowego

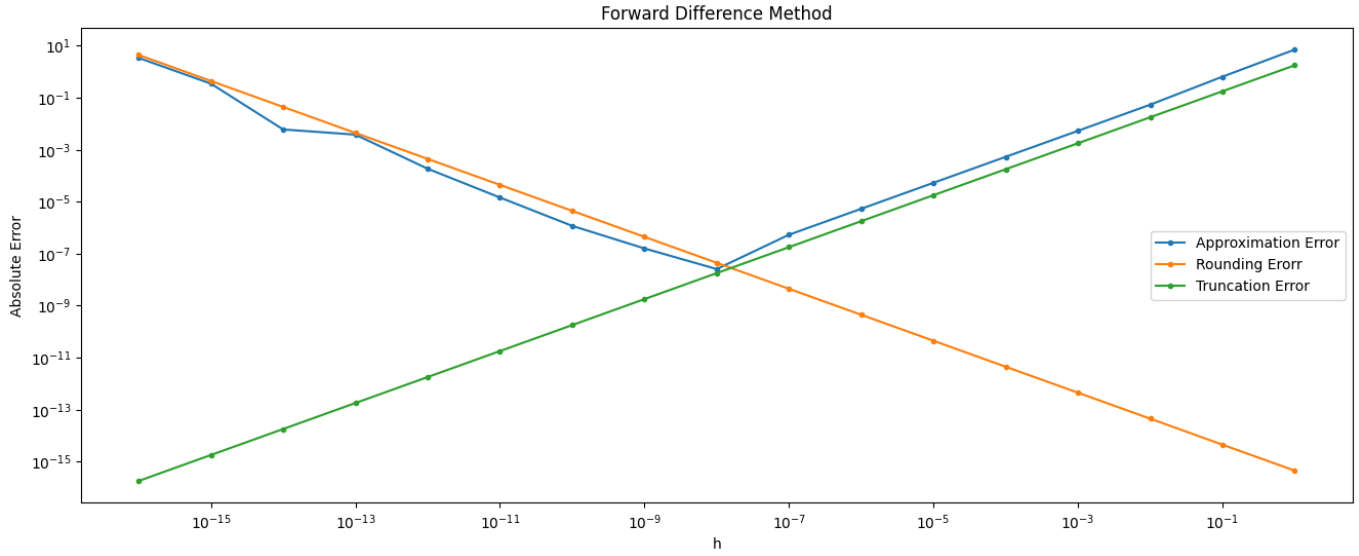
```
plt.figure(figsize=(10, 6))
plt.semilogy(range(n_fraction), error_fractions, label='Fractions')
plt.semilogy(range(n_single), error_single, label='Single Precision')
plt.semilogy(range(n_double), error_double, label='Double Precision')
plt.xlabel('k')
plt.ylabel('Relative Error')
plt.title('Relative Error w zależności od k')
plt.legend()
plt.show()
```

3. Wykresy, tabele, wyniki liczbowe

3.1 Zadanie 1

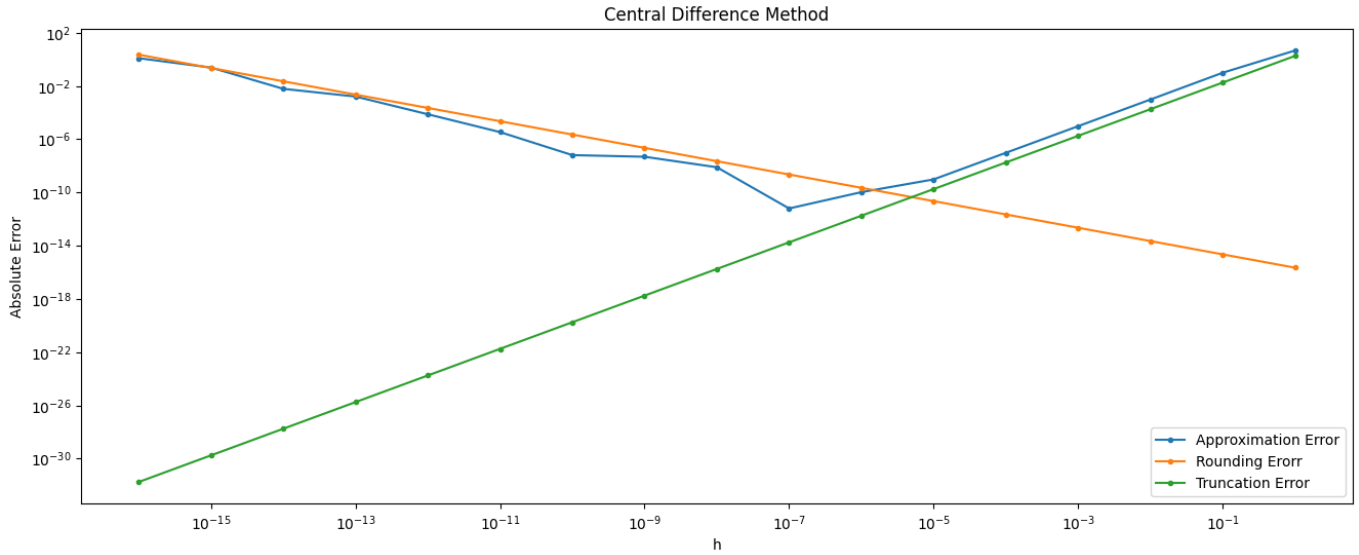
3.1.1

Wykres przedstawiający wartości błędu metody, błędu numerycznego oraz błędu obliczeniowego w zależności od wartości parametru h dla metody różnic skończonych.



3.1.2

Wykres przedstawiający wartości błędu metody, błędu numerycznego oraz błędu obliczeniowego w zależności od wartości parametru h dla metody różnic centralnych.



3.1.3

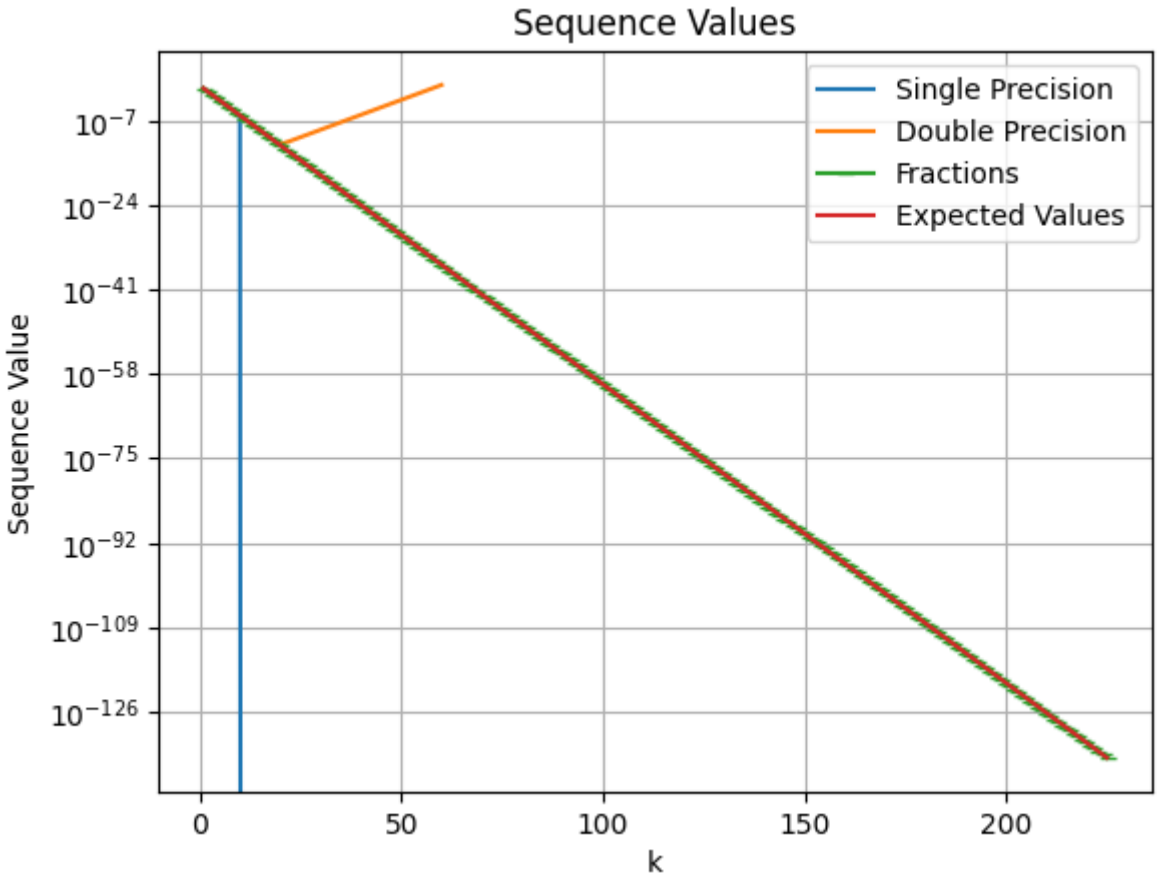
Zestawienie wartości h_{min} z minimum wartości błędu obliczeniowego ze względu na metodę

method	eperimental values	expected values
forward method	$2.55 \cdot 10^{-8}$	$9.12 \cdot 10^{-10}$
central method	$6.22 \cdot 10^{-13}$	$4.08 \cdot 10^{-6}$

3.2 Zadanie 2

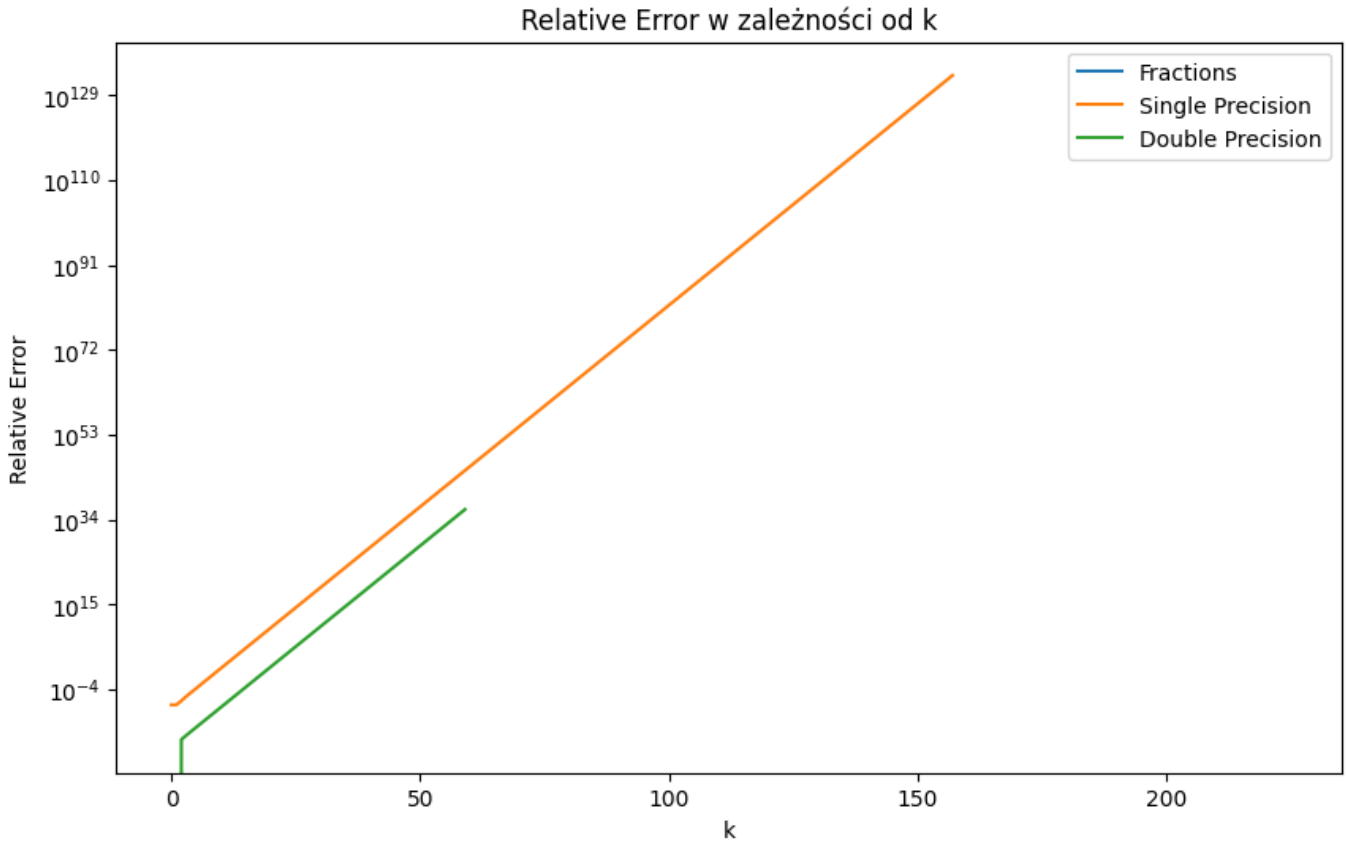
3.2.1

Wykres wartości ciągu w zależności od k



3.2.2

Wykres przedstawiający wartość bezwględną błędu względnego w zależności od k



4. Wnioski

W przypadku generowania poszczególnych wartości parametru h zauważono, że notacja numpy postaci:

```
h_values = 1/10**np.arange(0, 17)
```

nie zwracała poprawnych wyników w przypadku systemu operacyjnego windows. Dla potęg od -10 wzwyż wyniki były następujące:
$7.09 \cdot 10^{-10}, 8.22 \cdot 10^{-10}, -1.37 \cdot 10^{-9}, 7.59 \cdot 10^{-10}, 3.61 \cdot 10^{-9}, -6.53 \cdot 10^{-10}, 5.33 \cdot 10^{-10}$ dla odpowiednich $k \in [10, 16]$. Działo się tak z powodu braku specyfikacji typu *float* dla metody **arrange**.

4.1 Zadanie 1

Jak można zauważyć na wykresach w punktach 3.1.1 i 3.1.2 wraz ze wzrostem parametru h maleje błąd numeryczny, a rośnie błąd metody. Dla błędu metody różnic skończonych wartość błędu obliczeniowego jest najmniejsza dla $h = 10^{-8}$, a dla metody różnic centralnych dla $h = 10^{-7}$. Jednak warto zaznaczyć, że metoda różnic centralnych okazała się być obarczona mniejszymi wartościami bezwględnymi błędów niż metoda różnic skończonych. Porównując wartości minimalne błędów obliczeniowych z wartościami wyznaczonymi ze wzoru, można zauważyć, że w przypadku metody różnic skończonych, wartość wyznaczona wzorem była o jeden rząd wielkości mniejsza od wartości minimalnej. W przypadku metody różnic centralnych wartość wyznaczona wzorem była o sześć rzędów większa od wartości minimalnej.

4.2 Zadanie 2

W Zadaniu drugim, co widać na wykresie w punkcie 3.2.1, najlepszą reprezentacją okazała się **Frac** z biblioteki **Fractions**, jako że otrzymywane wartości pokrywały się z wartościami oczekiwanymi. W przypadku pojedyńczej precyzji w okolicach wartości rzędu 10^{-7} nastąpiło przekroczenie zakresu. W tym wypadku Python zamienia te wartości na bardzo małe, początkowo **-inf**, a następnie na **NaN**, stąd nagły spadek na wykresie. W przypadku podwójnej precyzji w okolicach wartości rzędu 10^{-12} zmieniła się monotoniczność ciągu. Dla tak małych wartości wystąpił skumulowany błąd zaokrągleń przez co otrzymane wartości nie pokrywały się z oczekiwanymi. Razem z odstającymi wartościami zwiększał się błąd względny reprezentacji pojedyńczej oraz podwójnej precyzji, co widać na wykresie w punkcie 3.2.2.

5 Bibliografia

https://en.wikipedia.org/wiki/IEEE_754

https://docs.python.org/3/library/fractions.html

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

https://pythonnumericalmethods.berkeley.edu/notebooks/chapter09.02-Floating-Point-Numbers.html