Artur Gęsiarz
Marzec 9, 2024

Laboratorium nr 1

MOwNiT – Analiza Błędów

1. Treść zadań
   1.1. Obliczyć przybliżoną wartość pochodnej funkcji, używając wzoru

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

Sprawdzić działanie programu dla funkcji $\tan(x)$ oraz dla $x = 1$.
Wyznaczyć błąd, porównując otrzymaną wartość numerycznej pochodnej z prawdziwą wartością.
Pomocna będzie tożsamość

$$tan'(x) = 1 + tan^2(x).$$

Na wspólnym rysunku przedstawić wykresy wartości bezwzględnej błędu metody, błędu numerycznego oraz błędu obliczeniowego w zależności od h dla $h = 10^{-k}, k = 0, \ldots, 16$. Użyć skali logarytmicznej na obu osiach. Odpowiedzieć na pytanie czy wykres wartości bezwzględnej błędu obliczeniowego posiada minimum.
Porównać wyznaczoną wartość h_{min} z wartością otrzymaną ze wzoru

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

Powtórzyć ćwiczenie używając wzoru różnic centralnych

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

Porównać wyznaczoną wartość h_{min} z wartością otrzymaną ze wzoru
$h_{min} \approx 2\sqrt[3]{3 * \epsilon\_mach/M}$, gdzie $M \approx |f'''(\mathrm{x})|$

   1.2. Napisać program generujący pierwsze n wyrazów ciągu zdefiniowanego równaniem różnicowym:

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

Wykonać obliczenia:
- używając pojedynczej precyzji oraz przyjmując n = 225
- używając podwójnej precyzji oraz przyjmując n = 60
- używając reprezentacji z biblioteki fractions oraz przyjmując n = 225.

Narysować wykres wartości ciągu w zależności od k. Użyć skali logarytmicznej na osi y. Następnie narysować wykres przedstawiający wartość bezwzględną błędu względnego w zależności od k.

Dokładne rozwiązanie równania różnicowego wynosi:

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

Odpowiedzieć na pytanie czy otrzymany wykres zachowuje się w ten sposób?

2. Rozwiązanie zadań

2.1. Rozwiązanie zadania pierwszego:

2.1.1. Implementacje pomocniczych funkcji:

```python
def function_tangens(x):
    """
    Funkcja oblicza tangensa z danego argumentu
    :param x: liczba rzeczywista
    """
    return np.tan(x)

def tangens_derivative(x):
    """
    Funkcja oblicza pochodna tangensa
    :param x: liczba rzeczywista
    :return: obliczona wartosc
    """
    return 1 + function_tangens(x) ** 2

def second_tangens_derivative(x):
    """
    Funkcja oblicza druga pochodna tangensa,
    po to aby obliczyc M
    :param x: liczba rzeczwista
    :return: wartosc drugiej pochodnej tangensa
    """
    return 2 * function_tangens(x) * tangens_derivative(x)

def third_tangens_derivative(x):
    """
    Funkcja oblicza trzecia pochodna tangensa,
    po to aby obliczyc M
    :param x: liczba rzeczwista
    :return: wartosc trzeciej pochodnej tangensa
    """
    return 2 * tangens_derivative(x) * (1 + function_tangens(x) ** 2) + \
        2 * function_tangens(x) * (2 * function_tangens(x) * tangens_derivative(x))
```

2.1.2. Implementacja funkcji przybliżających pochodne:

```python
def approx_derivative(x, h):
    """
    Funkcja oblicza przyblizona wartosc pochodnej funkcji tangensa
    :param x: liczba rzeczywista
    :param h: dany skok
    :return: przyblizona wartosc
    """
    return (function_tangens(x + h) - function_tangens(x)) / h

def approx_derivative_central(x, h):
    """
    Funkcja oblicza przyblizona wartosc pochodnej funkcji tangensa uzywajac wzoru roznic centralnych
    :param x:
    :param h:
    :return:
    """
    return (function_tangens(x + h) - function_tangens(x - h)) / ( 2 * h )
```

2.1.3. Implementacja funkcji obliczającej odpowiednio błąd numeryczny, metody oraz obliczeniowy:

```python
def numerical_error(h, epsilon):
    """
    Funkcja oblicza blad numeryczny
    :param epsilon:
    :param h:
    :return: blad numeryczny
    """
    return 2 * epsilon / h

def method_error(M, h):
    """
    Funkcja oblicza blad metody
    :param M:
    :param h:
    :return: blad metody
    """
    return M * h / 2

def computational_error(x, h, approx_function):
    """
    Funkcja oblicza blad obliczeniowy
    :param approx_function: wskaznik do funkcji, ktora przybliza pochodna funkcji tangensa
    :param x:
    :param h:
    :return: blad obliczeniowy
    """
    return np.abs(tangens_derivative(x) - approx_function(x, h))
```

2.1.4. Wyznaczanie błędów numerycznych, metody oraz obliczeniowych odpowiednio dla $h = 10^{-k}$, gdzie $k = 0, \dots, 16$ wykorzystując funkcje zaimplementowane powyżej.

2.1.4.1. Przybliżając funkcje pochodna tangensa, funkcją:

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

```
h_values = [10 ** -k for k in range(17)]
E_mach = np.finfo(float).eps
M = np.abs(second_tangens_derivative(1))

method_errors = [method_error(M, h) for h in h_values]
numerical_errors = [numerical_error(h, E_mach) for h in h_values]
computational_errors = [computational_error(1, h, approx_derivative) for h in h_values]

min_computational_error = min(computational_errors)
h_min_formula = 2 * np.sqrt(E_mach / M)
h_min_computational = h_values[computational_errors.index(min_computational_error)]
```

2.1.4.2. Przybliżając funkcje pochodna tangensa, funkcją:

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

```
h_values = [10 ** -k for k in range(17)]
E_mach = np.finfo(float).eps
M = np.abs(third_tangens_derivative(1))

method_errors = [method_error(M, h) for h in h_values]
numerical_errors = [numerical_error(h, E_mach) for h in h_values]
computational_errors = [computational_error(1, h, approx_derivative_central) for h in h_values]

min_computational_error = min(computational_errors)
h_min_formula = (3 * E_mach / M) ** (1/3)
h_min_computational = h_values[computational_errors.index(min_computational_error)]
```

2.1.5. Implementacja funkcji do rysowania wykresów

```
plt.figure(figsize=(10, 6))
plt.loglog(h_values, method_errors, label='Method Error')
plt.loglog(h_values, numerical_errors, label='Numerical Error')
plt.loglog(h_values, computational_errors, label='Computational Error')
plt.xlabel('h')
plt.ylabel('Absolute Error')
plt.title('Diagram of the analysis of calculation errors for the first method')
plt.legend()

plt.scatter(h_min_computational, min_computational_error, color='red', label=f'Minimum Computational
 Error = {h_min_computational:.2e}')
plt.axvline(x=h_min_formula, color='g', linestyle='--', label=f'h_min formula = {h_min_formula:.2e}')
plt.legend()

plt.show()
```

2.2. Rozwiązanie zadania drugiego:

2.2.1. Implementacja pomocniczych funkcji:

```python
def generate_sequence_single(n = 225):
    x = np.zeros(n, dtype=np.float32)
    x[0] = np.float32(1/3)
    x[1] = np.float32(1/12)
    for k in range(2, n):
        x[k] = np.float32(2.25) * x[k-1] - np.float32(0.5) * x[k-2]
    return x

def generate_sequence_double(n = 60):
    x = np.zeros(n, dtype=np.float64)
    x[0] = 1/3
    x[1] = 1/12
    for k in range(2, n):
        x[k] = 2.25 * x[k-1] - 0.5 * x[k-2]
    return x

def generate_sequence_fraction(n = 225):
    x = np.zeros(n, dtype=object)
    x[0] = Fraction(1,3)
    x[1] = Fraction(1,12)
    for k in range(2, n):
        x[k] = Fraction(9,4) * x[k-1] - Fraction(1,2) * x[k-2]
    return x
```

2.2.2. Implementacja funkcji reprezentującej faktyczną wartość ciągu

```python
def exact_solution(k):
    return 4 ** (-k) / 3
```

2.2.3. Implementacja funkcji do rysowania wykresu, porównującego działanie trzech funkcji z funkcją dającą prawidłowe wyniki

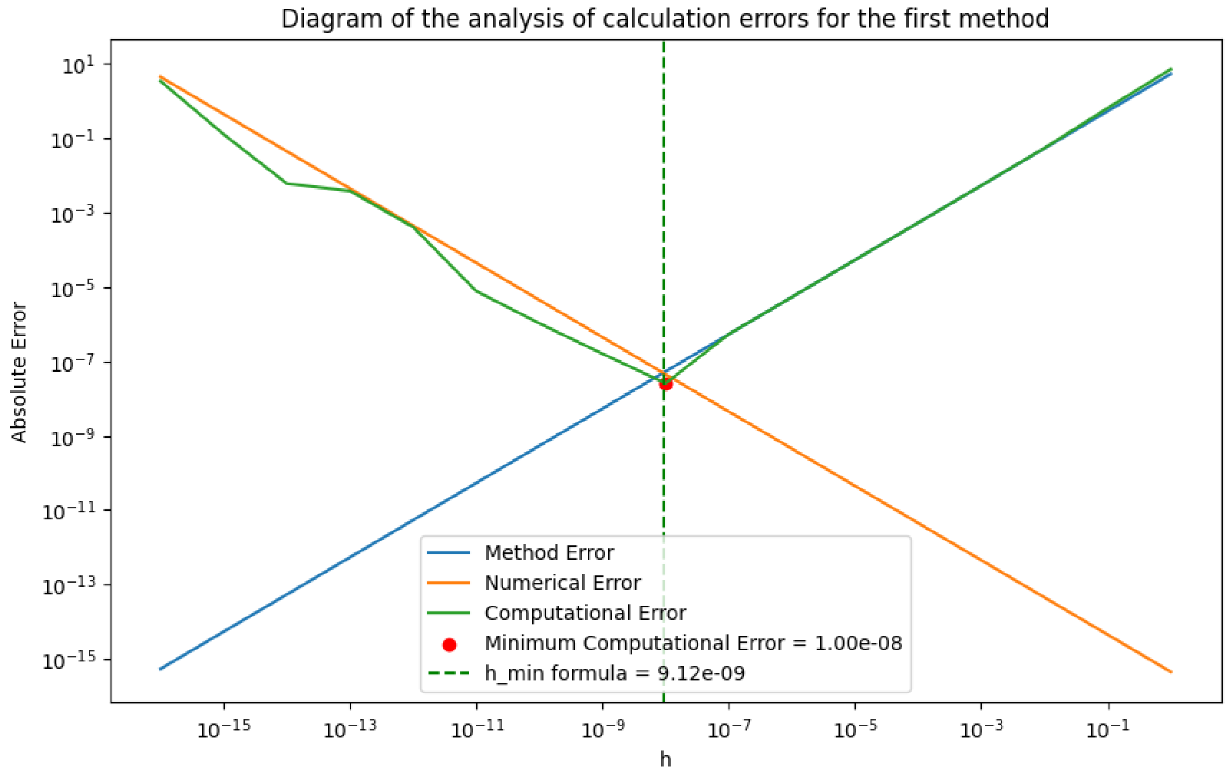
```
sequence_single = generate_sequence_single()
sequence_double = generate_sequence_double()
sequence_fraction = generate_sequence_fraction()
exact_values = [exact_solution(k) for k in range(2,225)]

plt.figure(figsize=(10, 6))
plt.loglog(range(225), sequence_single, label='Single Precision')
plt.loglog(range(60), sequence_double, label='Double Precision')
plt.loglog(range(225), sequence_fraction, label='Fractions')
plt.loglog(range(2,225), exact_values, label='Exact values', color="fuchsia")
plt.xlabel('k')
plt.ylabel('Value of Sequence')
plt.title('Comparison of sequence values from different functions')
plt.legend()
plt.grid(True)
plt.show()
```

2.2.4. Implementacja funkcji do rysowania wykresu, porównującego błędy bezwzględne otrzymane z trzech funkcji z różnymi precyzjami

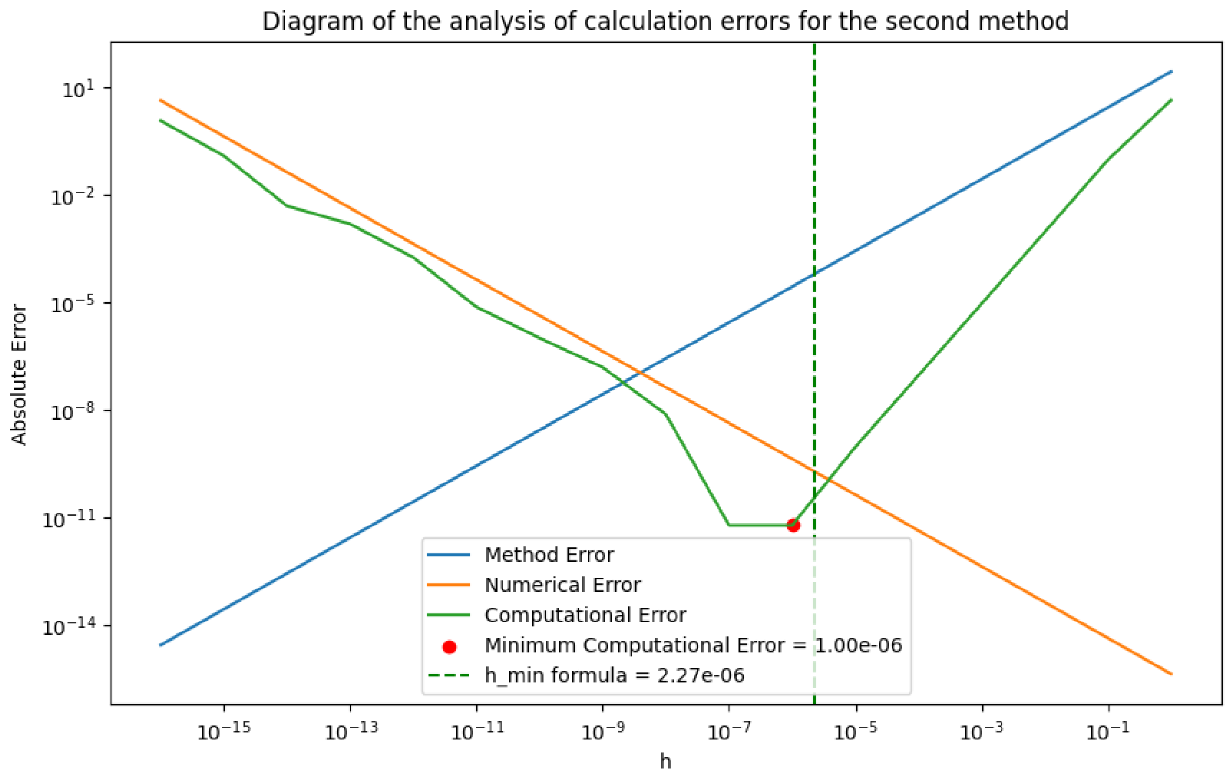
```
error_single = [absolute_relative_error(k, generate_sequence_single) for k in range(2,225)]
error_double = [absolute_relative_error(k, generate_sequence_double) for k in range(2,60)]
error_fraction = [absolute_relative_error(k, generate_sequence_fraction) for k in range(2,225)]

plt.figure(figsize=(10, 6))
plt.loglog(range(2,225), error_single, label='Error Single Precision')
plt.loglog(range(2,60), error_double, label='Error Double Precision')
plt.loglog(range(2,225), error_fraction, label='Error Fractions')
plt.xlabel('k')
plt.ylabel('Value of Sequence')
plt.title('Errors Sequence Values')
plt.legend()
plt.grid(True)
plt.show()
```

3. Wykresy

3.1. Wykresy do zadania pierwszego:

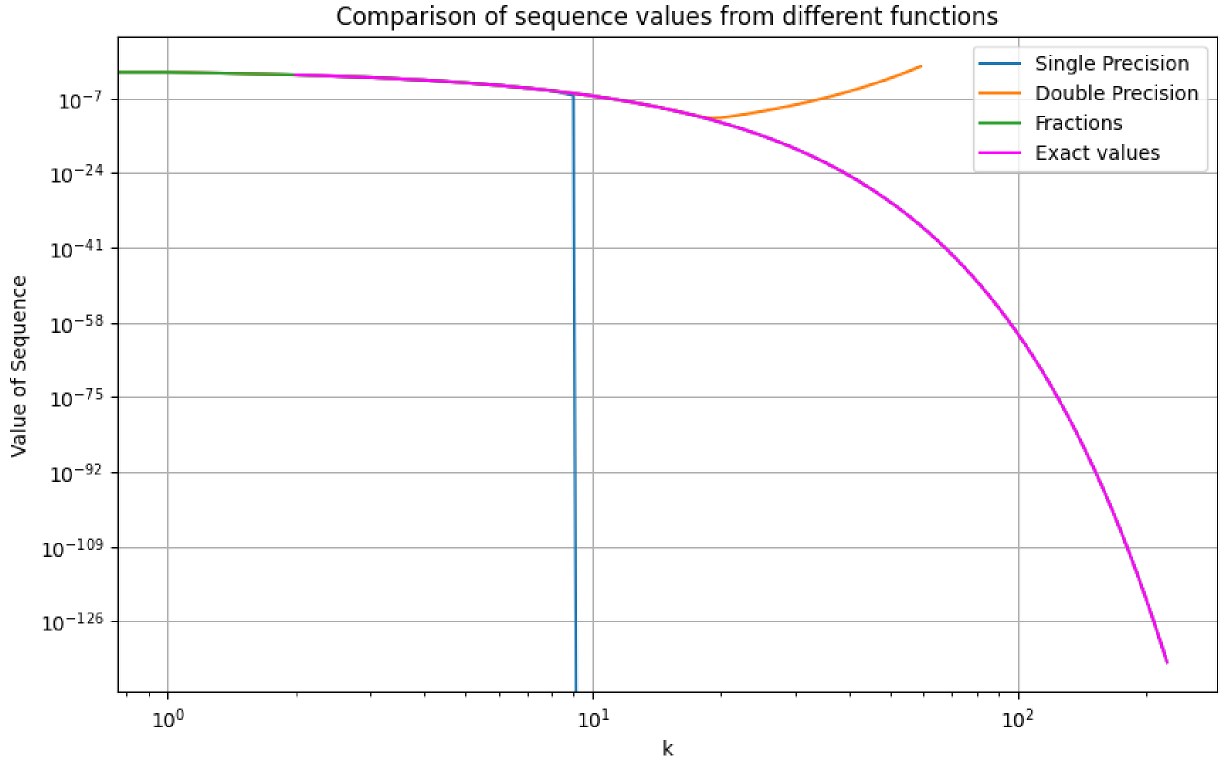


Wykres 1. Wykres opisujący poszczególne błędy w stosunku do h dla pierwszej metody

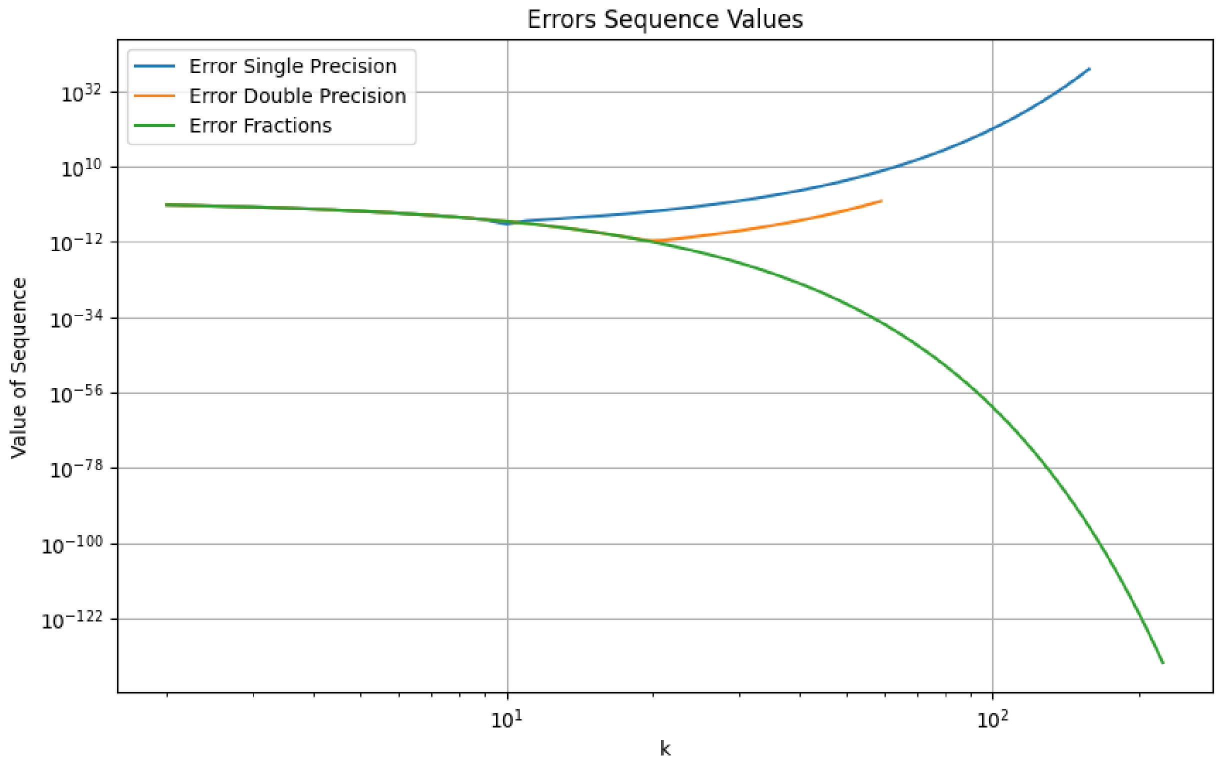


Wykres 2. Wykres opisujący poszczególne błędy w stosunku do h dla drugiej metody

3.2. Wykresy do zadania drugiego:



Wykres 3. Wykres opisujący wynik w zależności od trzech funkcji o różnych precyzjach z prawdziwą funkcją



Wykres 4. Wykres opisujący błąd bezwzględny względem trzech funkcji o różnych precyzjach

4. Tabele

4.1. Tabele dla zadania pierwszego

4.1.1. Tabela opisująca błąd obliczeniowy w stosunku do h dla pierwszej metody:

Wartość h	Błąd bezwzględny
1	7.1679
10^{-1}	0.648
10^{-2}	0.0543
10^{-3}	0.0053
10^{-4}	0.0005
10^{-5}	$5.33503 * 10^{-5}$
10^{-6}	$5.3348 * 10^{-6}$
10^{-7}	$5.3624 * 10^{-7}$
10^{-8}	$2.5541 * 10^{-8}$
10^{-9}	$1.5876 * 10^{-7}$
10^{-10}	$1.0469 * 10^{-6}$
10^{-11}	$7.7082 * 10^{-6}$
10^{-12}	0.0004
10^{-13}	0.0038
10^{-14}	0.0060
10^{-15}	0.1271
10^{-16}	3.4255

Tabela 1.

4.1.2. Tabela opisująca błąd obliczeniowy w stosunku do h dla drugiej metody:

Wartość h	Błąd bezwzględny
1	4.5180
10^{-1}	0.0974
10^{-2}	0.0009
10^{-3}	$9.4505 * 10^{-6}$
10^{-4}	$9.4503 * * 10^{-8}$
10^{-5}	$9.4991 * 10^{-10}$
10^{-6}	$6.2239 * 10^{-12}$
10^{-7}	$6.2239 * 10^{-12}$
10^{-8}	$7.7653 * 10^{-9}$
10^{-9}	$1.5876 * 10^{-7}$
10^{-10}	$1.0469 * 10^{-6}$
10^{-11}	$7.7082 * 10^{-6}$
10^{-12}	0.0001
10^{-13}	0.0015
10^{-14}	0.0060
10^{-15}	0.0050
10^{-16}	0.1271

Tabela 2.

4.2. Tabele dla zadania drugiego:

4.2.1. Tabela wyników dla każdej z trzech funkcji o różnych precyzjach w porównaniu do wyników funkcji rzeczywistej

k	Single Precision function	Double precision function	Fraction precision function	Exact function
20	-0.0016	$2.7717 * 10^{-12}$	$3.0316 * 10^{-13}$	$1.8947 * 10^{-14}$
40	-1755	$1.6348 * 10^{-6}$	$2.7572 * 10^{-25}$	$1.7232 * 10^{-26}$
60	-1840700400	1.7142	$2.5077 * 10^{-37}$	$1.5673 * 10^{-38}$
80	$-19301143 * 10^{9}$	1797558	$2.2807 * 10^{-49}$	$1.4254 * 10^{-50}$
100	$-2.023 * 10^{21}$	1884	$2.0743 * 10^{-61}$	$1.2964 * 10^{-62}$
120	$-2.122 * 10^{27}$	1884877076187	$1.8865 * 10^{-73}$	$1.1791 * 10^{-74}$
140	$-$	$-$	$1.7158 * 10^{-85}$	$1.0724 * 10^{-86}$
160	$-$	$-$	$1.5605 * 10^{-97}$	$9.7534 * 10^{-99}$
180	$-$	$-$	$1.4193 * 10^{-109}$	$8.8707 * 10^{-111}$
200	$-$	$-$	$1.2908 * 10^{-121}$	$8.06789 * 10^{-123}$
220	$-$	$-$	$1.1740 * 10^{-133}$	$7.3377 * 10^{-135}$

Tabela 3.

4.2.2. Tabela wyników dla każdej z trzech funkcji porównująca błąd bezwzględny

k	Single Precision function	Double precision function	Fraction precision function
20	0.00334	$2.4686 * 10^{-12}$	$5.6843 * 10^{-14}$
40	3510	$1.6348 * 10^{-6}$	$2.7572 * 10^{-25}$
60	3681400832	1.7142	$2.5077 * 10^{-37}$
80	3860228558815232	1797558	$4.2764 * 10^{-50}$
100	$4.04 * 10^{21}$	1884877076187	$3.8893 * 10^{-62}$

Tabela 4.

5. Wnioski

5.1. Wnioski dla pierwszego zadania:

Analizując wykres dla pierwszej metody (**Wykres 1.**) możemy zauważyć, że wykres błędu obliczeniowego posiada minimum lokalne wynoszące $\mathbf{2,55 * 10^{-8}}$, w punkcie $\mathbf{1 * 10^{-8}}$ a natomiast h_{min} wynosi $\mathbf{9.12 * 10^{-9}}$. Wyniki te różnią się o $\mathbf{8.76 * 10^{-10}}$.

Za to analizując wykres dla drugiej metody (**Wykres 2.**) możemy zauważyć, że wykres błędu obliczeniowego posiada minimum lokalne wynoszące $\mathbf{6,22 * 10^{-12}}$, w punkcie $\mathbf{1 * 10^{-6}}$, a h_{min} wynosi $\mathbf{2.27 * 10^{-6}}$. Wyniki w tym przypadku różnią się o $\mathbf{1.27 * 10^{-6}}$.

Jednak porównując te dwa wykresy (**Wykres 1., oraz Wykres 2.**) oraz odpowiadające im tabele wynikowe (**Tabela 1., Tabela 2.**) możemy zauważyć ze metoda druga jest znaczenie lepszą metodą obliczeniową. Jest to lepsza metoda, ponieważ otrzymujemy znacznie mniejszy błąd obliczeniowy, oraz minimum lokalne jest umieszczone znacznie niżej niż jest to w przypadku pierwszej metody – czyli druga metoda posiada znacznie mniejszy błąd obliczeniowy niż pierwsza.

5.2. Wnioski dla drugiego zadania:

Analizując wykres porównujący wszystkie 3 funkcje wyliczające tak naprawdę to samo, ale z różną precyzją porównanie z prawdziwym rozwiązaniem (**Wykres 3**), możemy zauważyć ze funkcja wykorzystująca precyzje z klasy ***Fraction,*** radzi sobie najlepiej, ponieważ pokrywa ona znaczna część prawdziwej funkcji, niż jak to jest w przypadku reszty funkcji. Dodatkowo na tym wykresie jest widoczny znaczy nagły spadek w dół funkcji wykorzystującej najmniejsza precyzje co prowadzi do późniejszych złych wyników.

Analizując dodatkowo wykres porównujący błędy, możemy również zauważyć tendencje funkcji wykorzystującej precyzje ***Fraction*** do największego spadku to znaczy ze jest najlepsza, a kiedy patrzymy na pozostałe funkcje o dziwo widzimy ze są one rosnące, a nie malejące. Mówi to nam ze, czym większe obliczamy wyrazy tym większy błąd otrzymujemy.
Kiedy spojrzymy na tabele przechowującej dokładne wyniki (**Tabela 3**. oraz **Tabela 4**), jest bardzo dobrze widoczne niedokładność funkcji wykorzystującej precyzje float32 jest to dla setnego wyrazu aż błąd $4.04 * 10^{21}$, kiedy to dla funkcji wykorzystującej precyzje ***Fraction*** jest to $3.8893 * 10^{-62}$.

5.3. Wnioski ogólne

Dlaczego klasa Fraction aż tak dobrze sobie radzi? - dzieje się tak ponieważ ona przechowuje liczby w postaci ułamków, co eliminuje błędy związane z reprezentacją liczb w formacie zmiennoprzecinkowym.

Liczby zmiennoprzecinkowe mają skończoną precyzję, niektóre operacje arytmetyczne mogą prowadzić do błędów zaokrąglenia, które się akumulują i prowadzą do niedokładnych wyników.

6. Bibliografia

Wykład MOwNiT - prowadzony przez dr. Inż. K. Rycerz
Prezentacje – dr. Inż. M. Kuta

7. Dodatkowe informacje

Rozwiązania obu tych zadań z dokładnym opisie znajdują się odpowiednio w plikach ex1.ipynb oraz ex2.ipynb.