Helena Szczepanowska, 29.02.2024

Laboratorium nr 1

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.

2. Kod

2.1 Zadanie 1

Celem zadania było numeryczne obliczenie pochodnej funkcji tan(x) na 2 sposoby i porównanie z jej rzeczywistą wartością dla x = 1 oraz $h = 10^{-k}$ $k = 0, ..., 16$. Do wykonania zadania skorzystano z pythonowych bibliotek `numpy` oraz `matplotlib.pyplot`

Do obliczenia pochodnych wykorzystane zostały następujące funkcje:

```
def f(x):
    return np.tan(x)

def f_prime(x):
```

```
    return 1 + np.tan(x)**2

def f_double_prime(x):
    return 2 * np.tan(x) * (1 + np.tan(x)**2)

def f_triple_prime(x):
    return 2 * (1 + np.tan(x)**2) * (1 + 3*np.tan(x)**2)

def numerical_derivative(x, h):
    return (f(x + h) - f(x)) / h

def central_difference(x, h):
    return (f(x + h) - f(x - h)) / (2 * h)
```

Za ich pomocą wykonano obliczenia wyliczające wartość pochodnej oraz błędu obliczeniowego:

```
i = 0
x0 = 1
for h in h_values:
    numerical_derivatives[i] = numerical_derivative(x0, h)
    numerical_derivatives_central[i] = central_difference(x0, h)
    true_derivative = f_prime(x0)
    computional_errors[i] = np.abs(numerical_derivatives[i] -
true_derivative)
    computional_errors_central[i] =
np.abs(numerical_derivatives_central[i] - true_derivative)
    i += 1
```

Do obliczenia błędu maszynowego epsilon skorzystano z funkcji finfo z biblioteki numpy, która oblicza maszynową precyzję epsilon dla typu danych zmiennoprzecinkowych float:

```
epsilon_machine = np.finfo(float).eps
```

Wyliczenie minimów i porównanie z faktyczną wartością podaną w treści zadania jest zrealizowane w następujący sposób:

```
M = np.abs(f_double_prime(x0))
M_central = np.abs(f_triple_prime(x0))

h_min = 2 * np.sqrt(epsilon_machine / M)
h_min_computational = min(computional_errors)
h_min_difference = np.abs(h_min - h_min_computational)

h_min_central = np.cbrt(3 * epsilon_machine / M_central)
h_min_computational_central = min(computional_errors_central)
```

```
h_min_difference_central = np.abs(h_min_central -
h_min_computational_central)
```

Dodatkowo wyliczone zostały błędy metody i numeryczne:

```
truncation_errors = 0.5 * M * h_values
rounding_errors = 2 * epsilon_machine / h_values

truncation_errors_central = (M_central * h_values**2)/6
rounding_errors = epsilon_machine / h_values
```

W celu analizy powyższe obliczenia przedstawione zostały na wykresach w skali logarytmicznej na obu osiach za pomocą funkcji loglog z biblioteki matplotlib.pyplot:

```
plt.loglog(h_values, computional_errors, label='Computional Error',
marker='o')
plt.loglog(h_values, truncation_errors, label='Truncation Error',
marker='o')
plt.loglog(h_values, rounding_errors, label='Rounding Error',
marker='o')

plt.loglog(h_values, computional_errors_central, label='Computional
Error', marker='o')
plt.loglog(h_values, truncation_errors_central, label='Truncation
Error', marker='o')
plt.loglog(h_values, rounding_errors, label='Rounding Error',
marker='o')
```

2.2 Zadanie 2

Celem zadania było obliczenie kolejnych wyrazów ciągu w sposób rekurencyjny, przy użyciu precyzji pojedynczej i podwójnej oraz notacji z biblioteki fractions i porównanie wyników z wyrazami obliczonymi wzorem ogólnym, a także obliczenie błędu względnego dla tych wyników. Do wykonania zadania skorzystano z pythonowych bibliotek `numpy`, `matplotlib.pyplot` oraz `fractions`.

Do wyliczenia wyrazów ciągu skorzystano z funkcji:

```
def sequence(n, precision):
    x = np.zeros(n, dtype=precision)
    x[0] = precision(1/3)
    x[1] = precision(1/12)
    for k in range(1, n-1):
        x[k+1] = precision(2.25 * x[k]) - precision(0.5 * x[k-1])

    return x
```

Istotne jest tutaj zachowanie precyzji dla każdego elementu równania poprzedzając je wyrażeniem, które konwertuje nam je na oczekiwaną precyzję.

Dla wyliczenia wyrazów ciągu korzystając z biblioteki fractions funkcja i ułamki zapisane są w danej postaci:

```
def fraction_sequence(n):
    x = [Fraction(0) for _ in range(n)]
    x[0] = Fraction(1, 3)
    x[1] = Fraction(1, 12)

    for k in range(1, n-1):
        x[k+1] = Fraction(9, 4) * x[k] - Fraction(1, 2) * x[k-1]

    return x
```

Do wyliczenia poprawnego rozwiązania użyto funkcji, która oblicza każdy kolejny wyraz za pomocą wzoru ogólnego:

```
def solution(n):
    solution = [0]*n
    for k in range(n):
        solution[k] = 4**(-k)/3

    return solution
```

Wartość bezwzględna błędu względnego dla każdej z precyzji została obliczona w następujący sposób:

```
errors_single = np.abs((x_single - solution_values_single) /
solution_values_single)
errors_double = np.abs((x_double - solution_values_double) /
solution_values_double)
errors_fraction = [abs((x - sol) / sol) for x, sol in zip(x_fraction,
solution_values_fraction)]
```

W celu analizy powyższe obliczenia przedstawione zostały na wykresach w skali logarytmicznej na osi y za pomocą funkcji semilogy z biblioteki matplotlib.pyplot:

Wykres wartości ciągu względem k:

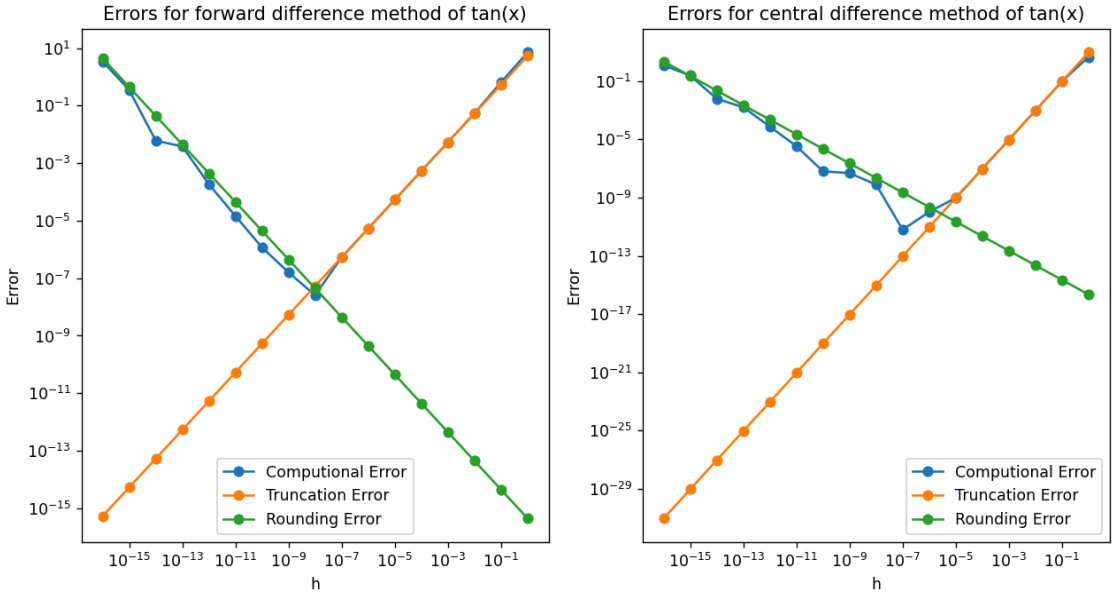
```
plt.semilogy(range(n_fraction), x_fraction, label='Fraction
Representation')
plt.semilogy(range(n_double), x_double, label='Double Precision')
plt.semilogy(range(n_single), x_single, label='Single Precision')
```

Wykres błędów względnych względem k:

```
plt.semilogy(range(n_fraction), errors_fraction, label='Fraction
Representation Error')
plt.semilogy(range(n_double), errors_double, label='Double Precision
Error')
plt.semilogy(range(n_single), errors_single, label='Single Precision
Error')
```

3. Wykresy, obliczenia

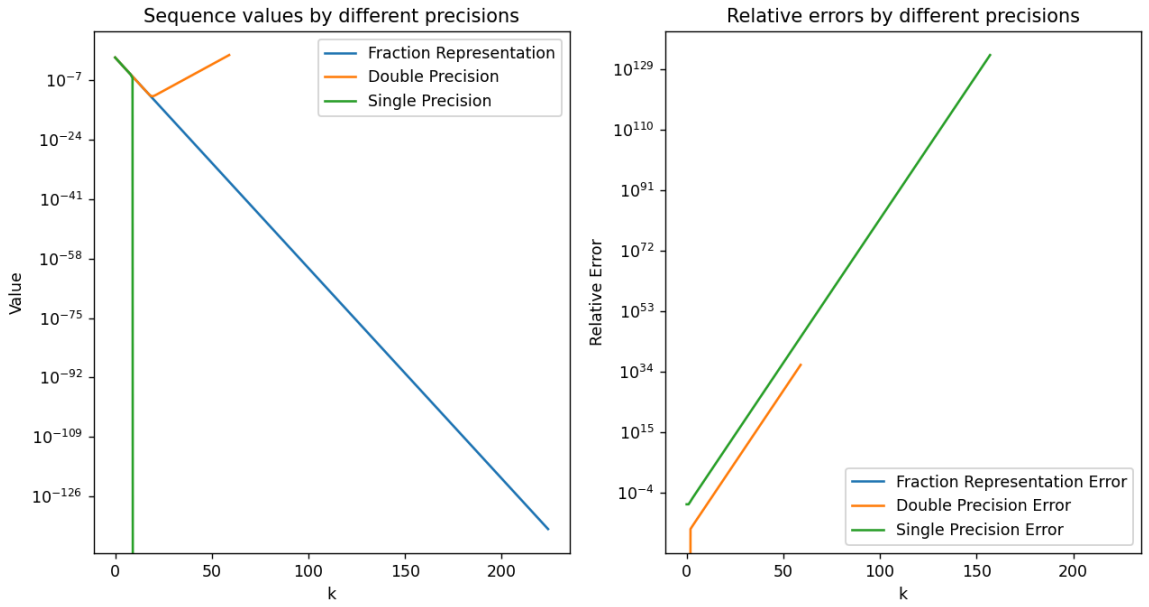
3.1 Zadanie 1



Difference of h_min in forward difference method 1.6417657363295778e-08

Difference of h_min in central difference method 2.273267933816964e-06

3.2 Zadanie 2



4. Wnioski

4.1 Zadanie 1

Analizując wykresy w obu przypadkach występują podobne trendy:

- błąd obliczeniowy (Computational Error) początkowo maleje w miarę zwiększania się h, osiągając minimum, po czym zaczyna rosnąć
- błąd metody (Truncation Error) stale rośnie w miarę zwiększania się h
- błąd numeryczny (Rounding Error) stale maleje w miarę zwiększania się h

Jednak warto zauważyć, że dla metody różnic centralnych błąd obliczeniowy maleje znacznie szybciej w porównaniu z metodą różnic do przodu i osiąga niższe wartości minimalne, zanim zacznie rosnąć, a błąd metody osiąga znacznie mniejsze wartości.

Z obu wykresów można wyciągnąć następujące wnioski:
Istnieje optymalna wartość kroku h dla każdej metody, przy której suma błędu obliczeniowego i błędu numerycznego jest minimalna. Wybierając zbyt duże h, dominować będzie błąd metody, podczas gdy zbyt małe h spowoduje wzrost błędu numerycznego. Metoda różnic centralnych jest w ogólności bardziej dokładna niż metoda różnic do przodu dla tych samych wartości h, ponieważ błędy są mniejsze w szerszym zakresie wartości h.

Dodatkowo analiza różnicy oczekiwanego minimum wykresu funkcji z rzeczywistym wskazuje na to, że wzory te dają przybliżenie wartości minimum, ale faktyczna optymalna wartość może się różnić, a wzór na minimum dla metody różnic do przodu oszacował je dokładniej.

4.2 Zadanie 2

Analizując wykres przedstawiający wartości kolejnych wyrazów ciągu dla różnych precyzji można zauważyć, że do pewnego małego k wszystkie stale maleją, jednak potem wartości dla każdej z nich zachowują się w inny sposób:

- dla pojedynczej precyzji wartości zaczynają znacznie odbiegać od spodziewanych co wskazuje na diametralne zwiększenie błędu numerycznego spowodowanego ograniczoną precyzją, dochodzi do błędu ponieważ wartość przekroczyła maksymalny zakres, jaki może być przechowany w precyzji pojedynczej
- dla podwójnej precyzji wartości kolejnych wyrazów ciągu od pewnego k zaczynają niespodziewanie rosnąć, wskazuje to na numeryczną niestabilność metody, ponieważ wartości ciągu powinny zbliżać się do zera
- dla obliczeń wykonanych przy pomocy biblioteki fractions wartości kolejnych wyrazów ciągu zachowują się w sposób właściwy i stale maleją

Dodatkowa analiza wykresu wartości bezwzględnej błędów względnych pozwala na wysnucie następujących wniosków:

- dla pojedynczej precyzji widoczny jest bardzo duży wzrost błędu numerycznego, jest to spodziewane, ponieważ pojedyncza precyzja ma ograniczoną liczbę cyfr znaczących, co powoduje, że błędy zaokrągleń rosną z każdym kolejnym obliczeniem rekurencyjnym
- dla podwójnej precyzji na początku wyniki są bardziej precyzyjne niż dla pojedynczej precyzji, ostatecznie również ulegają niestabilności i błąd dynamicznie wzrasta dla kolejnych k
- dla obliczeń wykonanych przy pomocy biblioteki fractions błąd nie występuje, co wskazuje na poprawne wyniki obliczeń

Biorąc pod uwagę powyższą analizę, możemy stwierdzić, że dla dużych wartości k, jedynie reprezentacja ułamkowa za pomocą biblioteki fractions daje wiarygodne wyniki, podczas gdy metody zmiennoprzecinkowe (pojedyncza i podwójna precyzja) stają się niestabilne numerycznie.

5. Bibliografia

- https://zingale.github.io/comp_astro_tutorial/basics/floating-point/numerical_error.html
- https://paulklein.ca/newsite/teaching/Notes_NumericalDifferentiation.pdf
- https://www.xilinx.com/applications/ai-inference/single-precision-vs-double-precision-main-differences.html