MOWNIT laboratorium 2

Hubert Miklas

Marzec 2025

1 Wstep

Laboratorium jest ciagiem dalszym laboratorium 1 które skupia sie na arytmetyce komputerowej.

2 Zadanie 1.

Celem zadania jest zaimplementowanie algorytmu obliczajacego funkcje wykładnicza e^x za pomoca nieskończonego szeregu Maclaurina. Szereg ten ma postać:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

W tym zadaniu rozważono kilka aspektów implementacji algorytmu, w tym kryterium zakończenia obliczeń, porównanie wyników z funkcja $exp(x)$ oraz pytanie o dokładność dla wartości $x < 0$.

2.1 Kryterium zakończenia obliczeń

Aby zakończyć obliczenia, przyjeto, że obliczenia zostana przerwane, gdy wartość kolejnego składnika szeregu osiagnie wartość mniejsza niż zadana dokładność ϵ. W kodzie implementujacym obliczenia, wartość ta została ustawiona na 1×10^{-15}.

2.2 Kod implementujacy obliczenia

```
from math import exp

def facts(n):
    if n == 0:
        return [1]
    results = [1]
    fact = 1
    for i in range(1, n):
        fact *= i
        results.append(fact)
    return results

def maclaurin_exp(x, epsilon=1e-15):
    fact_size = 10
    acc = 1
    fact = facts(fact_size)
    i = 0
    result = 0
    if x == 0:
        return 1
    while abs(acc / fact[i]) > epsilon:
        if i + 1 == fact_size:
            fact_size *= 2
            fact = facts(fact_size)
        result += acc / fact[i]
        acc *= x
        i += 1
    return result

def maclaurin_exp_negative(x, epsilon=1e-15):
    return 1 / maclaurin_exp(-x, epsilon)

def horner_maclaurin_exp(x, n=100, epsilon=1e-15):
    fact = facts(n)
    result = 1 / fact[n - 1]  # Initialize last term
    for i in range(n - 2, -1, -1):
        result = 1 / fact[i] + x * result
    return result
```

2.3 Wyniki obliczeń

Poniżej przedstawiono wyniki obliczeń dla różnych wartości x. Dla każdej wartości x obliczono wartość funkcji wykładniczej zarówno za pomoca zaimplementowanego algorytmu Maclaurina, jak i funkcji wbudowanej w biblioteke Python `exp()`. Porównano również wyniki obliczeń za pomoca algorytmu Hornera.

x	Moje obliczenia e^x	Biblioteka Python $\exp(x)$	Bład
-10	4.540×10^{-5}	4.540×10^{-5}	6.776×10^{-21}
-5	0.00674	0.00674	1.735×10^{-18}
-1	0.368	0.368	-5.551×10^{-17}
1	2.718	2.718	4.441×10^{-16}
5	148.413	148.413	-2.842×10^{-14}
10	22026.466	22026.466	-7.276×10^{-12}

2.4 Dokładność dla wartości $x < 0$

Przeprowadzono obliczenia dla $x < 0$ przy pomocy standardowego algorytmu oraz zmodyfikowanej wersji algorytmu, w której obliczenia sa wykonywane za pomoca e^{-x}, a nastepnie wynik jest odwrotnościa obliczonej wartości. Widać, że uzyskane wyniki sa zgodne z wartościami obliczonymi przy użyciu funkcji $\exp(x)$ z biblioteki Python. Błedy dla wartości $x < 0$ sa również minimalne i mieszcza sie w granicach dokładności ϵ.

2.5 Podsumowanie i wnioski na temat lepszego obliczania szeregu

Zaproponowany algorytm do obliczania funkcji wykładniczej e^x opiera sie na klasycznym szeregu Taylora:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

Dla wartości $x < 0$ bezpośrednie sumowanie szeregu może prowadzić do problemów numerycznych zwiazanych z sumowaniem wyrazów o zmiennych znakach, co zwieksza ryzyko utraty precyzji (tzw. *catastrophic cancellation*). Aby uzyskać dokładniejsze wyniki, zamiast bezpośrednio sumować wyrazy szeregu, wykorzystuje sie tożsamość:

$$e^x = \frac{1}{e^{-x}},$$

co pozwala na obliczenie szeregu dla e^{-x} (przy $x > 0$), w którym wszystkie składniki sa dodatnie. Nastepnie, poprzez wyliczenie odwrotności, uzyskuje sie wartość e^x. Takie przegrupowanie składników zmniejsza błedy zaokragleń i zwieksza stabilność numeryczna obliczeń [2, 1].

3 Zadanie 2

Celem zadania jest porównanie dokładności dwóch matematycznie równoważnych wyrażeń:

$$x^2 - y^2 \quad oraz \quad (x-y)(x+y)$$

w kontekście obliczeń arytmetyki zmiennoprzecinkowej. Oba wyrażenia sa matematycznie ekwiwalentne, jednak przy obliczeniach komputerowych moga wystapić różnice wynikajace z procesów zaokragleń.

3.1 Porównanie wyrażeń

Obliczenia przeprowadzone dla różnych wartości zmiennych x i y pokazuja, że bezpośrednie obliczenie $x^2 - y^2$ może prowadzić do utraty precyzji, szczególnie gdy x i y sa bardzo zbliżone, ponieważ nastepuje odejmowanie dwóch dużych liczb. Natomiast wyrażenie $(x - y)(x + y)$ najpierw wykonuje odejmowanie, a dopiero potem mnożenie, co redukuje wpływ błedów zaokragleń i czyni to wyrażenie numerycznie bardziej stabilnym [1].

3.2 Kod implementujacy obliczenia

```
def expression_1(x,y):
    return x**2 - y**2

def expression_2(x,y):
    return (x+y) * (x-y)

test_values = [
    (1e-14, 1e-14),
    (1e-14, -1e-14),
    (-1e-14, 1e-14),
    (-1e-14, -1e-14),
    (1.1e-14, 1e-14),
    (1e-14, 1.1e-14),
]

for x,y in test_values:
    expression_1_value = expression_1(x,y)
    expression_2_value = expression_2(x,y)
    print(f"Value from the x^2 - y^2 {expression_1_value}")
    print(f"Value from the (x + y)(x - y) {expression_2_value}")
    print(f"Difference between expr_1 and expr_2: {expression_1_value - expression_2_value}'
```

3.3 Wyniki obliczeń

Poniższa tabela przedstawia wyniki obliczeń dla przykładowych wartości zmiennych x i y:

x i y	Wyrażenie $x^2 - y^2$	Wyrażenie $(x+y)(x-y)$
$(1 \cdot 10^{-14}, 1 \cdot 10^{-14})$	0.0	0.0
$(1 \cdot 10^{-14}, -1 \cdot 10^{-14})$	0.0	0.0
$(-1 \cdot 10^{-14}, 1 \cdot 10^{-14})$	0.0	-0.0
$(-1 \cdot 10^{-14}, -1 \cdot 10^{-14})$	0.0	-0.0
$(1.1 \cdot 10^{-14}, 1 \cdot 10^{-14})$	$2.1000000000000007 \times 10^{-29}$	$2.100000000000001 \times 10^{-29}$
$(1 \cdot 10^{-14}, 1.1 \cdot 10^{-14})$	$-2.1000000000000007 \times 10^{-29}$	$-2.100000000000001 \times 10^{-29}$

3.4 Analiza wyników

Z wyników obliczeń wynika, że przy bardzo małych wartościach zmiennych (rzedu 10^{-14}) oba wyrażenia daja identyczne wyniki równe zeru w kontekście arytmetyki zmiennoprzecinkowej. Różnice w wynikach moga pojawiać sie przy bardziej zróżnicowanych wartościach zmiennych. W takich przypadkach, wyrażenie $(x-y)(x+y)$ okazuje sie być bardziej stabilne numerycznie, gdyż minimalizuje efekty błedów zaokragleń wynikajacych z odejmowania dużych, niemal równych wartości w wyrażeniu $x^2 - y^2$ [3].

3.5 Podsumowanie

W arytmetyce zmiennoprzecinkowej wyrażenie $(x-y)(x+y)$ jest zwykle obliczane z wieksza dokładnościa niż $x^2 - y^2$, szczególnie gdy wartości x i y sa bardzo zbliżone. Jest to zwiazane z mniejszym ryzykiem utraty precyzji przy odejmowaniu.

4 Zadanie 3

Celem zadania jest analiza dokładności obliczeń wyróżnika równania kwadratowego w zmiennoprzecinkowej arytmetyce przy użyciu znormalizowanego systemu zmiennoprzecinkowego. Rozważamy równanie kwadratowe w postaci:

$$ax^2 + bx + c = 0,$$

gdzie $a = 1.22$, $b = 3.34$ oraz $c = 2.28$. Obliczenia wykonujemy w systemie zmiennoprzecinkowym o podstawie $\beta = 10$ i dokładności $p = 3$ (czyli z trzema cyframi znaczacymi).

4.1 Podstawowe obliczenia

Wyróżnik równania kwadratowego wyraża sie wzorem:

$$\Delta = b^2 - 4ac.$$

Zadanie składa sie z nastepujacych kroków:

1. **(a)** Obliczenie wartości wyróżnika Δ w znormalizowanym systemie zmiennoprzecinkowym (z uwzglednieniem ograniczonej precyzji),

2. **(b)** Wyznaczenie dokładnej wartości wyróżnika w rzeczywistej arytmetyce,
3. **(c)** Oszacowanie wzglednego błedu obliczonej wartości wyróżnika.

Ze wzgledu na ograniczona precyzje (trzy cyfry znaczace), operacje arytmetyczne wykonywane na liczbach moga skutkować znaczaca utrata precyzji, szczególnie gdy składniki b^2 i $4ac$ sa do siebie bardzo zbliżone. W takim przypadku nawet niewielkie błedy zaokragleń moga wpłynać na końcowy wynik, co jest szczególnie istotne przy rozwiazywaniu równań kwadratowych.

4.2 Kod implementujacy obliczenia

```
    class Quadratic:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c

    def evaluate_simple(self,x):
        return x**2 * self.a + self.b * x + self.c

    def evaluate_horner(self,x):
        return self.c + x * ( self.b + x * self.a )

    def delta(self):
        return self.b ** 2 - 4*self.a*self.c


def main():
    a = 1.22
    b = 3.34
    c = 2.28
    q = Quadratic(a,b,c)
    delta = q.delta()
    print(f"The value of delta is {delta}")
    real_delta = 0.0292
    print(f"The real value of delta is {real_delta}")
    print(f"Relatice difference  between the real_delta and delta {abs(real_delta-delta)/rea

if __name__ == "__main__":
    main()
```

4.3 Analiza wyników

Analiza wyników obliczeń pozwala zauważyć, że:

- Obliczona wartość wyróżnika w systemie zmiennoprzecinkowym z ograniczona precyzja może znacznie odbiegać od dokładnej wartości obliczonej w arytmetyce rzeczywistej.
- W przypadku, gdy b^2 oraz $4ac$ sa do siebie bardzo zbliżone, wzgledny bład obliczeń wyróżnika staje sie bardzo duży.

4.4 Podsumowanie

Przeprowadzona analiza pokazuje, że w systemach o ograniczonej precyzji obliczeniowej należy szczególnie uważać przy operacjach, w których nastepuje odejmowanie dwóch niemal równych liczb. Zarówno przy obliczaniu szeregu wykładniczego dla $x < 0$, jak i przy wyliczaniu wyróżnika równania kwadratowego, zastosowanie metod minimalizujacych utrate precyzji (np. przegrupowanie składników lub odpowiednia reformulacja wyrażeń) może znaczaco poprawić wyniki obliczeń [2, 1].

References

[1] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[2] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[3] Katarzyna Rycerz. Wykład z przedmiotu metody obliczeniowe w nauce i technice. Akademia Górniczo-Hutnicza im. Stanisława Staszica, 2025.