Sprawozdanie z zadań z analizy numerycznej

Patryk Blacha, Radosław Szepielak

10 marca 2025

Spis treści

1 Zadanie 1: Obliczanie pochodnej funkcji

1.1 Wprowadzenie

W tym zadaniu obliczamy przybliżoną wartość pochodnej funkcji $f(x) = \tan(x)$ w punkcie $x_0 = 1.0$ przy użyciu dwóch metod: różnicy do przodu (forward difference) oraz różnicy centralnej (central difference). Następnie porównujemy wyniki z dokładną wartością pochodnej, korzystając z tożsamości:

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

1.2 Metoda różnicy do przodu

Wzór na różnicę do przodu jest następujący:

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

1.2.1 Błędy

W metodzie różnicy do przodu występują dwa główne źródła błędów:

- **Błąd metody (truncation error)**:

$$E_{\text{trunc}} \approx \frac{|f''(x)|}{2} \cdot h$$

- **Błąd obliczeniowy (roundoff error)**:

$$E_{\text{round}} \approx \frac{2\epsilon_{\text{mach}}}{h}$$

1.2.2 Optymalne h_{min}

Teoretycznie optymalna wartość h_{min} dla metody różnicy do przodu jest dana wzorem:

$$h_{\text{min}} \approx 2\sqrt{\frac{\epsilon_{\text{mach}}}{M}}$$

gdzie $M \approx |f''(x)|$.

1.3 Metoda różnicy centralnej

Wzór na różnicę centralną jest następujący:

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

1.3.1 Błędy

W metodzie różnicy centralnej błędy są następujące:

- **Błąd metody (truncation error)**:

$$E_{\text{trunc}} \approx \frac{|f'''(x)|}{6} \cdot h^2$$

- **Błąd obliczeniowy (roundoff error)**:

$$E_{\text{round}} \approx \frac{\epsilon_{\text{mach}}}{h}$$

1.3.2 Optymalne $h_{\textbf{min}}$

Teoretycznie optymalna wartość h_{min} dla metody różnicy centralnej jest dana wzorem:

$$h_{\text{min}} \approx \left(\frac{3\epsilon_{\text{mach}}}{M}\right)^{1/3}$$

gdzie $M \approx |f'''(x)|$.

1.4 Implementacja

Poniżej znajduje się implementacja obu metod w Pythonie:

1.4.1 Definicje funkcji

```
import numpy as np
import matplotlib.pyplot as plt

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

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

def f_second(x):
    return 2*(1/np.cos(x)**2)*np.tan(x)

def f_third(x):
    return 2*(1/np.cos(x)**2) + 6*(np.sin(x)**2)/(np.cos(x)**4)
```

1.4.2 Obliczenia

```
x0 = 1.0
d_true = f_analytical_prime(x0)
eps = np.finfo(float).eps
k_vals = np.arange(0, 17)
h_vals = 10.0**(-k_vals)
```

1.4.3 Różnice do przodu

```
d_forward = (f(x0 + h_vals) - f(x0)) / h_vals
error_forward = np.abs(d_forward - d_true)
E_trunc_forward = 0.5 * np.abs(f_second(x0)) * h_vals
E_round_forward = 2 * eps / h_vals

idx_min_forward = np.argmin(error_forward)
h_min_forward_emp = h_vals[idx_min_forward]
E_min_forward_emp = error_forward[idx_min_forward]

M_forward = np.abs(f_second(x0))
h_min_forward_theor = 2 * np.sqrt(eps / M_forward)
```

1.4.4 Różnice centralne

```
d_central = (f(x0 + h_vals) - f(x0 - h_vals)) / (2 * h_vals)
error_central = np.abs(d_central - d_true)
E_trunc_central = (np.abs(f_third(x0)) / 6) * h_vals**2
E_round_central = eps / h_vals

idx_min_central = np.argmin(error_central)
h_min_central_emp = h_vals[idx_min_central]
E_min_central_emp = error_central[idx_min_central]

M_central = np.abs(f_third(x0))
h_min_central_theor = (3 * eps / M_central)**(1/3)
```

1.5 Wyniki

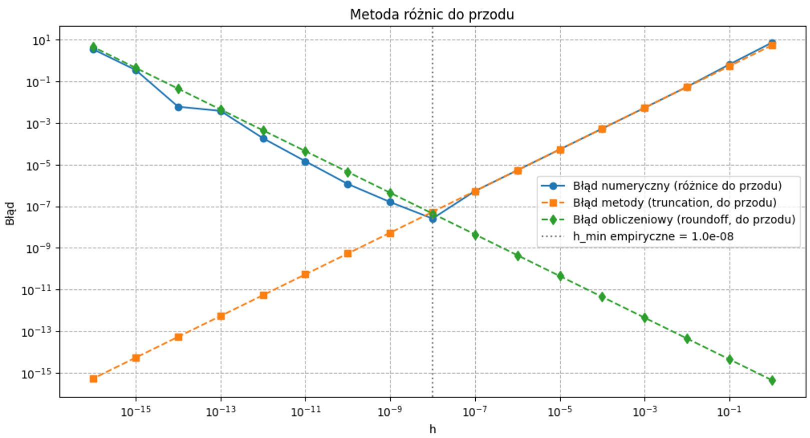
Poniżej przedstawiamy wyniki dla obu metod:

```
print("Dla metody r   nic do przodu:")
print(f"  Empiryczne h_min: {h_min_forward_emp:.3e}, E(h_min): {E_min_forward_emp:.3e}")
print(f"  Teoretyczne h_min (wz r (2)): {h_min_forward_theor:.3e}")
print()

print("Dla metody r   nic centralnych:")
print(f"  Empiryczne h_min: {h_min_central_emp:.3e}, E(h_min): {E_min_central_emp:.3e}")
print(f"  Teoretyczne h_min (wz r (4)): {h_min_central_theor:.3e}")
print()

if E_min_central_emp < E_min_forward_emp:
    print("Metoda r   nic centralnych jest dok adniejsza.")
else:
    print("Metoda r   nic do przodu jest dok adniejsza.")
```

1.6 Wykresy

Poniżej znajdują się wykresy błędów dla obu metod:

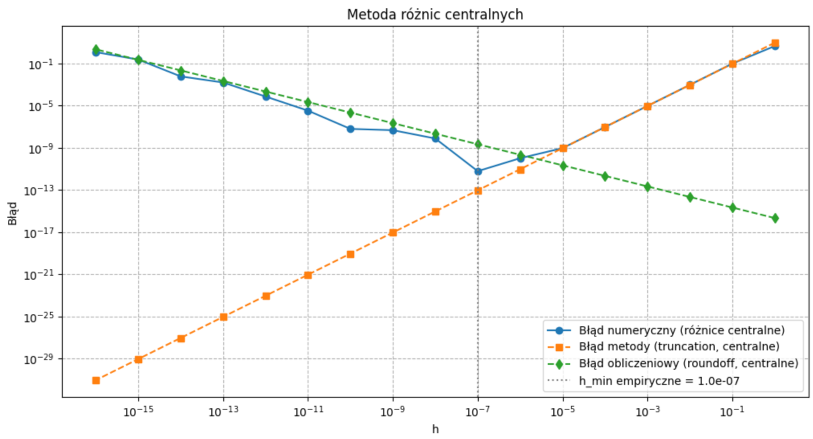
1.6.1 Metoda różnicy do przodu



Rysunek 1: Błędy dla metody różnicy do przodu

Empiryczne h_{min}: 1.000×10^{-8}, $E(h_{\text{min}})$: 2.554×10^{-8}
Teoretyczne h_{min} (wzór (1.2.2)): 9.124×10^{-9}

1.6.2 Metoda różnicy centralnej



Rysunek 2: Błędy dla metody różnicy centralnej

Empiryczne h_{min}: 1.000×10^{-7}, $E(h_{\text{min}})$: 6.223×10^{-12}
Teoretyczne h_{min} (wzór (1.3.2)): 2.273×10^{-6}

1.7 Wnioski

Na podstawie przeprowadzonych obliczeń i analizy błędów można stwierdzić, że metoda różnic centralnych jest dokładniejsza niż metoda różnic do przodu, co potwierdzają zarówno wyniki empiryczne, jak i teoretyczne.

2 Zadanie 2: Sumowanie liczb zmiennoprzecinkowych

2.1 Wstęp

Celem zadania było zbadanie wpływu różnych metod sumowania na dokładność obliczeń przy użyciu liczb zmiennoprzecinkowych pojedynczej precyzji. W szczególności, porównano pięć różnych metod sumowania:

- (a) Sumowanie w kolejności generowania z akumulatorem podwójnej precyzji,
- (b) Sumowanie w kolejności generowania z akumulatorem pojedynczej precyzji,
- (c) Sumowanie z użyciem algorytmu Kahana,
- (d) Sumowanie w porządku rosnącym,
- (e) Sumowanie w porządku malejącym.

Dla każdej metody obliczono względny błąd sumowania w zależności od liczby elementów $n = 10^k$, gdzie $k = 4, 5, 6, 7, 8$. Jako wartość referencyjną przyjęto sumę obliczoną za pomocą funkcji `math.fsum`, która zapewnia wysoką precyzję.

2.2 Implementacja

Poniżej znajduje się implementacja w Pythonie:

2.2.1 Definicje funkcji

```
import numpy as np
import matplotlib.pyplot as plt
import math

# Funkcja realizuj ca sumowanie Kahana w pojedynczej precyzji
def kahan_sum(x):
    suma = np.float32(0.0)
    komp = np.float32(0.0)
    for xi in x:
        y = np.float32(xi - komp)
        temp = np.float32(suma + y)
        komp = np.float32((temp - suma) - y)
        suma = temp
    return suma
```

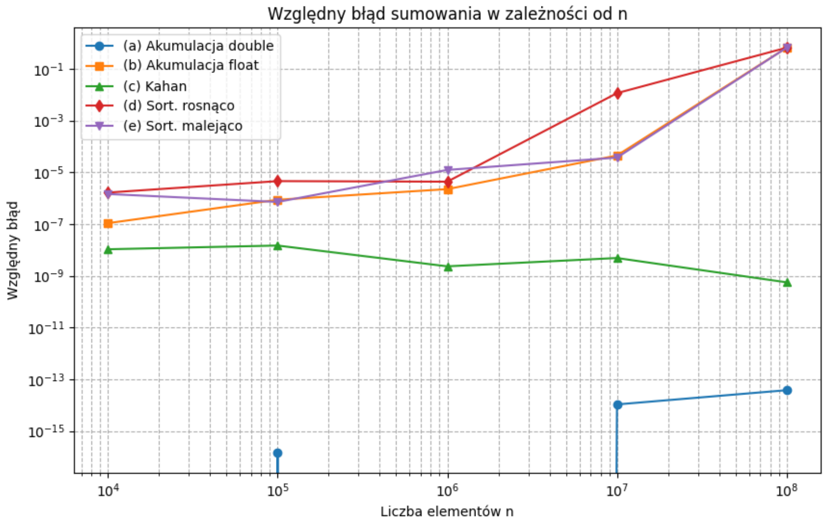
2.2.2 Obliczenia

```
# Lista warto ci n: n = 10^k, k = 4,5,6,7,8
n_values = [10 ** k for k in range(4, 9)]

# Listy do przechowywania wzgl dnych b  d w dla poszczeg lnych metod
errors_a = []  # (a) podw jna precyzja
errors_b = []  # (b) pojedyncza precyzja
errors_c = []  # (c) Kahan
errors_d = []  # (d) sortowanie rosn co
errors_e = []  # (e) sortowanie malej co

for n in n_values:
    # Generujemy n liczb pojedynczej precyzji
    x = np.random.uniform(0, 1, n).astype(np.float32)
    # Prawdziwa suma (wysoka precyzja)   wykorzystujemy math.fsum
    true_sum = math.fsum(x.tolist())

    # (a) Sumowanie w kolejno ci generowania z akumulatorem podw jnej precyzji
    sum_a = np.cumsum(x, dtype=np.float64)[-1]

    # (b) Sumowanie w kolejno ci generowania z akumulatorem pojedynczej precyzji
    sum_b = np.cumsum(x, dtype=np.float32)[-1]

    # (c) Sumowanie Kahana   algorytm sumowania z kompensacj , akumulacja w
    pojedynczej precyzji
    sum_c = kahan_sum(x)

    # (d) Sumowanie w kolejno ci rosn cej (od najmniejszych do najwi kszych)
    x_sorted_asc = np.sort(x)
    sum_d = np.cumsum(x_sorted_asc, dtype=np.float32)[-1]

    # (e) Sumowanie w kolejno ci malej cej (od najwi kszych do najmniejszych)
    x_sorted_desc = np.sort(x)[::-1]
    sum_e = np.cumsum(x_sorted_desc, dtype=np.float32)[-1]

    # Obliczamy wzgl dny b  d dla ka dej metody: |suma_metody - true_sum| / |
    true_sum|
    err_a = abs(sum_a - true_sum) / abs(true_sum)
    err_b = abs(sum_b - true_sum) / abs(true_sum)
    err_c = abs(sum_c - true_sum) / abs(true_sum)
    err_d = abs(sum_d - true_sum) / abs(true_sum)
    err_e = abs(sum_e - true_sum) / abs(true_sum)

    errors_a.append(err_a)
    errors_b.append(err_b)
    errors_c.append(err_c)
    errors_d.append(err_d)
    errors_e.append(err_e)

    print(f"n = {n:>10}:")
    print(f"  (a) podw jna precyzja: suma = {sum_a:.8e}, b  d = {err_a:.8e}")
    print(f"  (b) pojedyncza precyzja: suma = {sum_b:.8e}, b  d = {err_b:.8e}")
    print(f"  (c) Kahan:             suma = {sum_c:.8e}, b  d = {err_c:.8e}")
    print(f"  (d) sort. rosn co:     suma = {sum_d:.8e}, b  d = {err_d:.8e}")
```

```
    print(f"  (e) sort. malej co:    suma = {sum_e:.8e}, b   d = {err_e:.8e}")
    print()
```

2.3 Wykresy

Poniżej znajduje się wykres względnego błędu w zależności od liczby elementów n:



Rysunek 3: Względny błąd sumowania w zależności od liczby elementów n dla różnych metod sumowania.

2.4 Dyskusja

- Metoda (a) z użyciem akumulatora podwójnej precyzji charakteryzuje się najmniejszym błędem, co wynika z większej precyzji obliczeń.
- Metoda (b) z użyciem akumulatora pojedynczej precyzji wykazuje większy błąd niż metoda (a), co jest spowodowane ograniczoną precyzją liczb pojedynczej precyzji.
- Algorytm Kahana (metoda c) pozwala na znaczące zmniejszenie błędu w porównaniu do standardowego sumowania w pojedynczej precyzji, dzięki kompensacji błędów zaokrągleń.
- Sumowanie w porządku rosnącym (metoda d) daje lepsze wyniki niż sumowanie w porządku malejącym (metoda e), co jest zgodne z oczekiwaniami, ponieważ sumowanie od najmniejszych do największych wartości minimalizuje błędy zaokrągleń.

2.5 Wnioski

- Użycie akumulatora podwójnej precyzji (metoda a) jest najbardziej efektywnym sposobem na zmniejszenie błędu sumowania.
- Algorytm Kahana (metoda c) jest skutecznym narzędziem do poprawy dokładności sumowania w pojedynczej precyzji.
- Sumowanie w porządku rosnącym (metoda d) jest lepsze niż sumowanie w porządku malejącym (metoda e) pod względem dokładności.

```
n =      10000:
  (a) podwójna precyzja: suma = 4.98390337e+03, błąd = 0.00000000e+00
  (b) pojedyncza precyzja: suma = 4.98390283e+03, błąd = 1.08598304e-07
  (c) Kahan:             suma = 4.98390332e+03, błąd = 1.06266509e-08
  (d) sort. rosnąco:     suma = 4.98391162e+03, błąd = 1.65489144e-06
  (e) sort. malejąco:    suma = 4.98391064e+03, błąd = 1.45894814e-06

n =     100000:
  (a) podwójna precyzja: suma = 5.00377078e+04, błąd = 1.45409491e-16
  (b) pojedyncza precyzja: suma = 5.00376641e+04, błąd = 8.73525330e-07
  (c) Kahan:             suma = 5.00377070e+04, błąd = 1.47979435e-08
  (d) sort. rosnąco:     suma = 5.00379375e+04, błąd = 4.59110349e-06
  (e) sort. malejąco:    suma = 5.00376719e+04, błąd = 7.17393078e-07

n =    1000000:
  (a) podwójna precyzja: suma = 5.00160064e+05, błąd = 0.00000000e+00
  (b) pojedyncza precyzja: suma = 5.00161188e+05, błąd = 2.24695365e-06
  (c) Kahan:             suma = 5.00160062e+05, błąd = 2.32629676e-09
  (d) sort. rosnąco:     suma = 5.00162250e+05, błąd = 4.37127359e-06
  (e) sort. malejąco:    suma = 5.00153844e+05, błąd = 1.24358460e-05
```

Rysunek 4: Dane z terminala

```
n =    1000000:
  (a) podwójna precyzja: suma = 5.00160064e+05, błąd = 0.00000000e+00
  (b) pojedyncza precyzja: suma = 5.00161188e+05, błąd = 2.24695365e-06
  (c) Kahan:             suma = 5.00160062e+05, błąd = 2.32629676e-09
  (d) sort. rosnąco:     suma = 5.00162250e+05, błąd = 4.37127359e-06
  (e) sort. malejąco:    suma = 5.00153844e+05, błąd = 1.24358460e-05

n =   10000000:
  (a) podwójna precyzja: suma = 4.99973502e+06, błąd = 1.06176400e-14
  (b) pojedyncza precyzja: suma = 4.99996000e+06, błąd = 4.49975141e-05
  (c) Kahan:             suma = 4.99973500e+06, błąd = 4.87079915e-09
  (d) sort. rosnąco:     suma = 5.05767200e+06, błąd = 1.15880092e-02
  (e) sort. malejąco:    suma = 4.99954850e+06, błąd = 3.73068476e-05

n =  100000000:
  (a) podwójna precyzja: suma = 4.99970960e+07, błąd = 3.77021275e-14
  (b) pojedyncza precyzja: suma = 1.67772160e+07, błąd = 6.64436191e-01
  (c) Kahan:             suma = 4.99970960e+07, błąd = 5.58941790e-10
  (d) sort. rosnąco:     suma = 1.67772160e+07, błąd = 6.64436191e-01
  (e) sort. malejąco:    suma = 1.67772160e+07, błąd = 6.64436191e-01
```

Rysunek 5: Dane z terminala

3 Zadanie 3: Stabilność numeryczna wyrażeń

3.1 Rozwiązania

3.1.1 (a) $\sqrt{x+1}-1$, $x \approx 0$

Aby uniknąć zjawiska kancelacji, mnożymy przez sprzężenie:

$$\sqrt{x+1}-1 = \frac{(\sqrt{x+1}-1)(\sqrt{x+1}+1)}{\sqrt{x+1}+1} = \frac{x}{\sqrt{x+1}+1}.$$

3.1.2 (b) $x^2 - y^2$, $x \approx y$

Różnicę kwadratów zapisujemy jako iloczyn:

$$x^2 - y^2 = (x - y)(x + y).$$

Dzięki temu eliminujemy problem z odejmowaniem bliskich wartości.

3.1.3 (c) $1 - \cos x$, $x \approx 0$

Stosujemy tożsamość trygonometryczną:

$$1 - \cos x = 2 \sin^2 \frac{x}{2}.$$

3.1.4 (d) $\cos^2 x - \sin^2 x$, $x \approx \frac{\pi}{4}$

Korzystamy z tożsamości:

$$\cos^2 x - \sin^2 x = \cos 2x.$$

Dla $x \approx \frac{\pi}{4}$, mamy $\cos 2x = -\sin(2(x - \frac{\pi}{4}))$, co jest bardziej stabilne numerycznie.

3.1.5 (e) $\ln x - 1$, $x \approx e$

Przekształcamy wyrażenie:

$$\ln x - 1 = \ln \frac{x}{e}.$$

Dzięki temu unikamy odejmowania wartości bliskich sobie.

3.1.6 (f) $e^x - e^{-x}$, $x \approx 0$

Używamy funkcji hiperbolicznej:

$$e^x - e^{-x} = 2 \sinh x.$$

Dla małych x rozwijamy $\sinh x$ w szereg Taylora:

$$\sinh x = x + \frac{x^3}{3!} + \mathcal{O}(x^5).$$

To pozwala na stabilniejsze obliczenia numeryczne.

4 Zadanie 4: Porównanie sprawności kolektorów słonecznych S1 i S2

4.1 Wstęp

Efektywność kolektora słonecznego dana jest wzorem:

$$\eta = \frac{K\,Q\,T_d}{I},$$

gdzie:

- K – stała znana z dużą dokładnością,
- Q – objętość przepływu,
- T_d – różnica temperatur,
- I – natężenie promieniowania.

Dla dwóch kolektorów obliczono:

$$\eta_{S1} = 0.76 \quad \text{oraz} \quad \eta_{S2} = 0.70.$$

4.2 Błędy pomiarowe

Wielkości Q, T_d oraz I zmierzono z następującymi względnymi błędami:

Wielkość	$S1$	$S2$
Q	1.5%	0.5%
T_d	1.0%	1.0%
I	3.6%	2.0%

4.3 Propagacja błędu

Zakładając, że K jest znane bardzo dokładnie, względny błąd efektywności η wyraża się przy pomocy zasady dla iloczynu i ilorazu:

$$\frac{|\Delta\eta|}{\eta} \approx \frac{|\Delta Q|}{Q} + \frac{|\Delta T_d|}{T_d} + \frac{|\Delta I|}{I}.$$

Kolektor S1

Dla S1:

$$\frac{\Delta\eta_{S1}}{\eta_{S1}} = 1.5\% + 1.0\% + 3.6\% = 6.1\%.$$

Błąd bezwzględny wynosi:

$$\Delta\eta_{S1} = 0.76 \times 0.061 \approx 0.046.$$

Przyjmując, że rzeczywista efektywność mieści się w przedziale:

$$\eta_{S1} \in [0.76 - 0.046,\ 0.76 + 0.046] \approx [0.714,\ 0.806].$$

Kolektor S2

Dla S2:

$$\frac{\Delta\eta_{S2}}{\eta_{S2}} = 0.5\% + 1.0\% + 2.0\% = 3.5\%.$$

Błąd bezwzględny wynosi:

$$\Delta\eta_{S2} = 0.70 \times 0.035 \approx 0.0245.$$

Przyjmując przedział niepewności:

$$\eta_{S2} \in [0.70 - 0.0245,\ 0.70 + 0.0245] \approx [0.6755,\ 0.7245].$$

4.4 Analiza porównawcza

Dla kolektora S1 efektywność mieści się w przedziale [0.714, 0.806], natomiast dla S2 w przedziale [0.6755, 0.7245]. Zauważamy, że:

- Górna granica dla S2 wynosi około 0.7245,
- Dolna granica dla S1 wynosi około 0.714.

Przedziały te częściowo się nakładają, co oznacza, że w wyniku błędów pomiarowych nie możemy z całą pewnością stwierdzić, że rzeczywista efektywność S1 jest większa niż S2.

4.5 Wniosek

Pomimo nominalnie wyższej sprawności $S1$ (0.76) w porównaniu do $S2$ (0.70), niepewności pomiarowe (szczególnie większy błąd w I dla S1) powodują, że przedziały niepewności dla obu kolektorów się nakładają. Dlatego na podstawie dostępnych danych nie możemy być pewni, że kolektor $S1$ ma większą sprawność niż kolektor $S2$.