Sprawozdanie

Obliczenia naukowe

Lista 1

Piotr Gałwiaczek

Numer indeksu: 221495

1. Zadanie 1

1.1. Epsilon maszynowy

1.1.1. Opis problemu

Wyznaczenie epsilona maszynowego w sposób iteracyjny dla typów *Float16*, *Float32*, oraz *Float64*.

1.1.2. Rozwiązanie

Tworzę pętlę, która w każdej iteracji dzieli przez dwa, dopóki 1+x ≤ 1. Po skończeniu pętli ostatnia liczba to epsilon maszynowy.

```
1.  Inicjuj zmienne x i y wartością 1.0 odpowiedniego typu
2.      Dopóki 1 + x > 1
3.          y = x
4.          x = x/2
5.      return y
```

Rysunek 1 Kod

1.2. Liczba eta

1.2.1. Opis problemu

Szybkie wyznaczenie liczby eta

1.2.2. Rozwiązanie

Podobnie jak w zadaniu 1.1 dzielę przez 2, aż warunek start/2 >0 nie zostanie spełniony.

```
1.  Inicjuj zmienna start wartoscia 1.0 odpowiedniego typu
2.  while start/2 > 0
3.      Dziel start przez 2
4.  return start
```

Rysunek 2 Kod

1.3. Liczba MAX

1.3.1. Opis problemu

Wyznaczenie *maksymalnej liczby* jaką może przyjąć dana arytmetyka zmiennopozycyjna.

1.3.2. Rozwiązanie

Wyznaczając liczbę *MAX* posługuję się zmienną *x* w której znajdzie się owa maksymalna wartość, oraz *ile_dodajemy*, której wartość jest dodawana do x. Na początku w pętli dodajemy do *x* w każdej *i*-tej iteracji wartość $ile\_dodajemy=2^i$. W momencie gdy wartość $x+ile\_dodajemy=\infty$ wychodzimy z pętli.

```
1.  function count_max(typ)
2.      x = typ(1)
3.      ile_dodajemy = typ(2)
4.      while x+ile_dodajemy nie wyszło poza zakres
5.          Dodaj w i-tej iteracji do start 2^i
6.      while dodanie do x ile_dodajemy nie zmieniło wyniku
7.          while x+ile_dodajemy wychodzi poza zakres
8.              Dziel ile_dodajemy przez 2
9.          oldValue = x
10.         while x+ile_dodajemy nie wyszło poza zakres
11.             oldValue = x
12.             x += ile_dodajemy
13.             if oldValue == x
14.                 Wyjdź z głównej pętli
15.     return x
```

Rysunek 3 Kod

W następnej pętli na początku każdej iteracji szukamy liczby której dodanie do x nie spowoduje osiągnięcia przez nią wartości ∞. Liczbę tą szukamy dzieląc zmienna *ile_dodajemy* przez 2. Gdy dana liczba zostanie odnaleziona, dodajemy jej wartość do *x*. Jeżeli dodanie zmiennej *ile_dodajemy* nie wpływa na wartość *x*, oznacza to, że otrzymaliśmy już maksymalną liczbę w danej arytmetyce i kończymy procedure, w przeciwnym razie dodajemy wartość zmiennej *ile_dodajemy* ponownie. W momencie gdy suma tych zmiennych będzie równa ∞ kończymy sumowanie i rozpoczynamy kolejną iterację.

Po wyjściu z pętli otrzymana wartość *x* jest wartością *MAX*.

1.4. Wyniki oraz ich interpretacja

	Float16	Float32	Float64
Obliczone macheps	0.000977	1.1920929f-7	2.220446049250313e-16
Systemowe eps	0.000977	1.1920929f-7	2.220446049250313e-16
Macheps w float.h	-	1.192093e-07	2.22045e-016
Obliczone eta	6.0e-8	1.0f-45	5.0e-324
Systemowe eta	6.0e-8	1.0f-45	5.0e-324
Obliczone MAX	6.55e4	3.4028235f38	1.7976931348623157e308
Systemowe realmax	6.55e4	3.4028235f38	1.7976931348623157e308
MAX w float.h	-	3.402823e+038	1.797693e+308

Wyniki otrzymane z napisanych funkcji oraz z odpowiadających im funkcji wbudowanych pokrywają się. Oznacza to, że kody z *rysunków 1,2*, oraz *3* prawidłowo wyznaczyły wartości *epsilona maszynowego*, liczby *eta* i *MAX* dla poszczególnych typów zmiennopozycyjnych.
Liczba *macheps* jest równa podwojonej precyzji arytmetyki, czyli *macheps=2ε*, gdzie $\varepsilon=\frac{1}{2}\beta^{1-t}$, a *t* to liczba bitów mantysy.
MIN_{sub} to najmniejsza liczba nieznormalizowana większa od 0 jaką możemy reprezentować za pomocą danej arytmetyki zmiennopozycyjnej. W standardzie IEE-754 liczby zmiennopozycyjne są normalizowane, chyba że są one bardzo małe. *Subnormal numbers* to liczby z nieznormalizowaną mantysą i z najmniejszą możliwą cechą (u nas reprezentowaną przez same zera). Eta jest najmniejszą taką liczbą, zatem MIN_{sub} i eta są tym samym.

2. Zadanie 2 – Wyrażenie Kahana

2.1. Opis problemu

W tym zadaniu należy sprawdzić, czy wyrażenie *3*(*$\frac{4}{3}$ *– 1)-1* obliczone w odpowiedniej arytmetyce zmiennopozycyjnej daje wartość równą *epsilonowi maszynowemu*.

2.2. Rozwiązanie

Obliczam wyrażenie dla trzech typów zmiennopozycyjnych.

2.3. Wyniki oraz ich interpretacja

	Float16	Float32	Float64
Obliczone wyrażenie	-0.000977	1.1920929f-7	-2.220446049250313e-16
Systemowe eps	0.000977	1.1920929f-7	2.220446049250313e-16

Wyniki pokrywają się, lecz dla typów *Float16* oraz *Float64* są one z przeciwnym znakiem.

3. Zadanie 3 – równomierne rozmieszczenie liczb w [1,2]

3.1. Opis problemu

W tym zadaniu należy sprawdzić jak rozmieszczone są liczby zmiennopozycyjne w przedziale [1,2], $[\frac{1}{2}, 1], oraz\ [2, 4]$.

3.2. Rozwiązanie

Tworzę funkcję, która dostaje na wejściu odpowiedni typ danych, ilość liczb które chcemy wyświetlić, początek przedziału, oraz liczbę δ która oznaczać będzie różnicę pomiędzy kolejno wypisywanymi liczbami.

3.3. Wyniki oraz ich interpretacja

Przedział	δ
$[\frac{1}{2}, 1]$	2^{-53}
[1,2]	2^{-52}
[2,4]	2^{-51}

Dla liczb z przedziału [1,2] różnica pomiędzy kolejnymi liczbami wynosi $\delta = 2^{-52}$. Dzieje się tak ponieważ standard *IEEE 754* zapewnia „ukrycie" pierwszego bitu będącego 1, gdzie w naszym przypadku jest to bit odpowiedzialny za część całkowitą liczby. W związku z tym na zapisanie części ułamkowej pozostają nam 52 bity.

Dla liczb w przedziale [2,4] liczby są rozmieszczone co $\delta = 2^{-51}$. Jest to wynikiem tego, że do zapisania liczb w tym przedziale należy przesunąć bity o jedną pozycję w prawo. W związku z tym tracimy jeden bit części ułamkowej.

Liczby w przedziale $[\frac{1}{2}, 1]$ powstały przez przesunięcie mantysy o jeden w lewo, by ukryć pierwszy bit oznaczony 1. Dzięki temu $\delta = 2^{-53}$.

4. Zadanie 4 – najmniejsza liczba 1<x<2 taka, że x*(1/x)≠1

4.1. Opis problemu

W tym zadaniu mamy znaleźć najmniejszą liczbę *x* w danej arytmetyce, która nie spełnia równania *x*(1/x)≠1*.

4.2. Rozwiązanie

Do rozwiązania tego zadania utworzyliśmy pętle rozpoczynającą działanie dla liczby *x = 1*. W każdej iteracji wartość tej liczby jest zwiększana o $\delta = 2^{-52}$, przez co nie ominiemy żadnej cyfry w przedziale [1,2]. Pętla kończy działanie w momencie gdy wartość owego wyrażenia nie będzie spełniona.

4.3. Wyniki oraz ich interpretacja

1. Liczba x dla ktorej x*(1/x)!=1 to: 1.000000057228997
2. Wynik ten wynosi: 0.9999999999999999

Rysunek 4 Wynik przedstawiający szukającą najmniejszą liczbę spełniającą równanie

Ten nieprzewidywalny wynik jest konsekwencją wykonania najpierw działania *(1/x)* które poprzez specyfikę arytmetyki może dawać niedokładny wynik. Następnie wymnożenie tego wyniku przez *x* daje nam inna liczbę niż się tego spodziewamy.

5. Zadanie 5 – iloczyn skalarny wektorów

5.1. Opis problemu

W tym zadaniu mamy za zadanie obliczyć iloczyn skalarny dla tablic 5-cio elementowych:

- *x = [2.718281828, −3.141592654, 1.414213562, 0.5772156649, 0.3010299957]*
- *y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049]*

Mamy to wykonać na cztery sposoby:

a) od elementów pierwszych w tablicy, do ostatnich
b) od elementów ostatnich z tablicy, do pierwszych
c) w kolejności od największego do najmniejszego elementu tablicy
d) w kolejności od najmniejszego do największego elementu tablicy

5.2. Rozwiązanie

Do wykonania tego zadania utworzyliśmy funkcje otrzymujące jako parametry tablice, obliczające iloczyn skalarny w odpowiedniej kolejności, oraz wypisujące wynik na ekranie.

5.3. Wyniki oraz ich interpretacja

Kolejność sumowania	Float32	Float64
W przód	-0.4999443	1.0251881368296672e-10
W tył	-0.4543457	-1.5643308870494366e-10
Od najmniejszego do największego	-0.5	0.0
Od największego do najmniejszego	-0.5	0.0

Jak widać wyniki się różnią. Wynika to z tego, że w momencie gdy dodajemy do siebie liczby z których jedna z nich jest dużo większa od drugiej, wyrównanie cech powoduje zatracenie wielu danych mniejszej liczby. Dlatego w podpunktach c) oraz d) po wykonaniu mnożenia otrzymujemy sumy częściowe o wartościach bardzo odległych od siebie. Wiemy, że te wyniki nie mogą być poprawne, gdyż redukcja cyfr przy sumowaniu może okazać się bardzo duża. Wyniki sumowania w przód lub w tył okazują się być najbliższe prawdziwemu, lecz nadal nie mamy pewności który z nich jest bardziej poprawny. Dzieje się tak, ponieważ zadanie obliczenia iloczynu skalarnego jest źle uwarunkowane.

6. Zadanie 6 – obliczenie wyrażeń dwóch funkcji

6.1. Opis problemu

W zadaniu 6 mamy do obliczenia wartości dwóch, pozornie jednakowych funkcji $f(x)=\sqrt{x^2+1}-1$ oraz $g(x)=\frac{x^2}{\sqrt{x^2+1}+1}$. Należy zbadać wiarygodność wyniku obu z nich.

6.2. Rozwiązanie

Do wykonania tego zadania utworzyliśmy dwie funkcje: pierwsza zwracająca wartość funkcji *f(x),* druga *g(x).* Rozwiązania są generowane w pętli począwszy od i=1 do i=n. W każdej iteracji na ekranie wypisywane są wartości $f(8^{-i})$ oraz $g(8^{-i})$.

6.3. Wyniki oraz ich interpretacja

i	f(x)	g(x)
1	0.0077822185373186414	0.0077822185373187065
2	0.00012206286282867573	0.00012206286282875901
3	1.9073468138230965e-6	1.907346813826566e-6
4	2.9802321943606103e-8	2.9802321943606116e-8
5	4.656612873077393e-10	4.6566128719931904e-10
6	7.275957614183426e-12	7.275957614156956e-12
7	1.1368683772161603e-13	1.1368683772160957e-13
8	1.7763568394002505e-15	1.7763568394002489e-15
9	0.0	2.7755575615628914e-17
10	0.0	4.336808689942018e-19
11	0.0	6.776263578034403e-21
12	0.0	1.0587911840678754e-22
13	0.0	1.6543612251060553e-24
14	0.0	2.5849394142282115e-26
15	0.0	4.0389678347315804e-28
16	0.0	6.310887241768095e-30
17	0.0	9.860761315262648e-32
18	0.0	1.5407439555097887e-33
19	0.0	2.407412430484045e-35
20	0.0	3.76158192263132e-37

Jak widać wyniki do pewnego momentu prawie się pokrywają, lecz dla *i≥-9* jest pomiędzy nimi ogromna różnica. Dzieje się tak, ponieważ odejmowanie liczb bliskich siebie daje niepoprawne wyniki. W podanym przykładzie gdy w każdej iteracji zmniejszamy x wartość wyrażenia $\sqrt{x^2+1}$ będzie coraz bliższa 1. Po odjęciu od niej 1 wyrażenie będzie generowało niepoprawny wynik, czyli 0. Aby temu zapobiec, warto wyrażenie przekształcić do postaci w której unikamy odejmowania liczb bliskich siebie. Przykładem takiego wyrażenia jest funkcja g(x), dlatego daje ona bardziej wiarygodne wyniki.

7. Zadanie 7 – przybliżona wartość pochodnej w punkcie

7.1. Opis problemu

W tym zadaniu należy obliczyć przybliżoną wartość pochodnej z podanego wzoru, oraz porównać ją z prawdziwą wartością pochodnej w tym punkcie.

7.2. Rozwiązanie

W zadaniu tym utworzyliśmy dwie funkcje:

- $f(x) = \sin(x) + \cos(3x)$
- $g(x) = \cos(x) – 3\sin(3x)$

gdzie $f'(x) = g(x)$. Następnie w każdej i-tej iteracji pętli dla $h = 2^{-i}$, oraz z wartością x podaną w parametrze funkcji sprawdzaliśmy:

- przybliżoną wartość funkcji f(x)
- wartość błędu, czyli $|f(x)-g(x)|$
- wartość wyrażenia 1+h

Wyniki oraz ich interpretacja

Na początku iteracji wyniki dosyć znacząco się różnią – błąd jest duży. Sytuacja poprawia się w późniejszej fazie doświadczenia, lecz dla bardzo małego h znów otrzymujemy bardzo niekorzystne wyniki.