Obliczenia Naukowe
Lista 1

Yuliia Melnyk
246202

23 października 2022

1 Zadanie I

W zadaniu 1 należało wyznaczyć maszynowe epsilon, etę i licabę maksymalną dla arytmetyk `Float16`, `Float32` i `Float64`

1.1 Epsilon maszynowy

Epsilonem maszynowym (*macheps*) nazywamy taką najmniejszą liczbę $macheps > 0$, że $fl(1.0 + macheps) > 1.0$, gdzie fl oznacza arytmetykę, w której chcemy wykonać powyższe działanie.

1.1.1 Problem

W zadaniu należało, przy użyciu języka `Julia`, wyznaczyć metodą iteracyjną wartość *macheps* dla zadanych arytmetyk (`Float16`, `Float32`, `Float64`) oraz porównać otrzymane wyniki z wartościami zwracanymi przez funkcję `eps`.

1.1.2 Rozwiązanie problemu

Wybieramy liczbę którą będziemy zmniejszać, została wybrana jedynka odpowiedniego typu `Float`. Zmniejszamy jedynkę dopóki $fl(1.0 + liczba) > 1.0$. Ostatnia taka liczba będzie naszym wynikiem.

1.1.3 Wyniki

Dla poszczególnych typów arytmetyk zostały uzyskane następujące wyniki.

Arytmetyka `Float16`
Wartość uzyskana w wyniku działania algorytmu: 0.000977
Wartość domyślna dla arytmetyki `Float16`: 0.000977
Wartość dla języka `C` w pliku nagłówkowym `float.h`: *nie zdefiniowano*

Arytmetyka `Float32`
Wartość uzyskana w wyniku działania algorytmu: $1.1920929 \cdot 10^{-7}$
Wartość domyślna dla arytmetyki `Float32`: $1.1920929 \cdot 10^{-7}$
Wartość dla języka `C` w pliku nagłówkowym `float.h`: $1.1920929 \cdot 10^{-7}$

Arytmetyka `Float64`
Wartość uzyskana w wyniku działania algorytmu: $2.220446049250313 \cdot 10^{-16}$
Wartość domyślna dla arytmetyki `Float64`: $2.220446049250313 \cdot 10^{-16}$
Wartość dla języka `C` w pliku nagłówkowym `float.h`: $2.220446049250313 \cdot 10^{-16}$

1.1.4 Wnioski

Wyniki otrzymane przez algorytm są zgodne ze zwracanymi przez funkcję `eps`. Wraz ze wzrostem precyzji arytmetyki maleje wartość *macheps*. Zmniejszanie się wartości wynika z faktu, że w przypadku arytmetyki o większej precyzji możemy zapisać więcej cyfr znaczących, przez to rzadziej występuje zjawisko "ucinania bitów" i zaokrąglania. Występuje ścisły związek liczby *macheps* z precyzją arytmetyki - określa ona najmniejszą wartość, która dodana do dowolnej, większej liczby wpływa jeszcze na obliczenia. Wartości *macheps* dla odpowiednich arytmetyk języka `C` zawarte w pliku `float.h` są takie same jak te dla języka `Julia`.

1.2 Eta

Liczbą *eta* nazywamy taką liczbę, że $eta > 0.0$ w zadanej arytmetyce, czyli jest to najmniejsza możliwa liczba większa od wartości 0.0.

1.2.1 Problem

W zadaniu należało, przy użyciu języka `Julia`, wyznaczyć metodą iteracyjną liczbę *eta* dla zadanych arytmetyk (`Float16`, `Float32`, `Float64`) oraz porównać otrzymane wyniki z wartościami zwracanymi przez funkcję `nextfloat` oraz znaleźć związek otrzymanych wyników z wartością MIN_{sub}.

1.2.2 Rozwiązanie problemu

Bierzemy jedynkę odpowiedniego typu `Float` i zmniejszamy tą liczbę dopóki $liczba > 0.0$

1.2.3 Wyniki

W wyniku działania programu otrzymano następujące wyniki dla poszczególnych arytmetyk:

Arytmetyka `Float16`
Wartość uzyskana w wyniku działania algorytmu: $6.0 \cdot 10^{-8}$
Wartość domyślna dla arytmetyki `Float16`: $6.0 \cdot 10^{-8}$

Arytmetyka `Float32`
Wartość uzyskana w wyniku działania algorytmu: $1.0 \cdot 10^{-45}$
Wartość domyślna dla arytmetyki `Float32`: $1.0 \cdot 10^{-45}$

Arytmetyka `Float64`
Wartość uzyskana w wyniku działania algorytmu: $5.0 \cdot 10^{-324}$
Wartość domyślna dla arytmetyki `Float64`: $5.0 \cdot 10^{-324}$

1.2.4 Wnioski

Wartości *eta* obliczone przez algorytm są takie same, jak te zwracane przez funkcję `nextfloat`. Wynika to z faktu, że z powodu zwiększonej długości mantysy możemy reprezentować mniejsze liczby, stąd pierwsza możliwa do zapisania liczba większa od 0.0 jest mniejsza. Jest to najmniejsza możliwa do zapisania liczba większa od 0.0. Liczba *eta* ma ścisły związek z liczbą MIN_{sub}. Jej reprezentacja bitowa dla odpowiednich arytmetyk wygląda następująco:

Arytmetyka `Float32`: "00000000000000000000000000000001"
Arytmetyka `Float64`: "0000000000000000000000000000000000000000000000000000000000000001"

Jak widać, wszystkie bity cechy to 0. Jest to zatem liczba zdenormalizowana (*subnormal*).

1.3 MAX

Liczba MAX to największa możliwa do uzyskania wartość w zadanej arytmetyce zmiennopozycyjnej.

1.3.1 Problem

W zadaniu należało, przy użyciu języka `Julia`, wyznaczyć metodą iteracyjną wartość MAX dla zadanych arytmetyk (`Float16`, `Float32`, `Float64`) oraz porównać wyniki z wartościami zwracanymi przez funkcję `realmax` oraz danymi zwartymi w pliku nagłówkowym `float.h` dla języka `C`.

1.3.2 Rozwiązanie problemu

Przy pomocy funkcji `isinf()` możemy sprawdzić czy liczba należy do arytmetyki. Liczba może być uważana za ∞ kiedy jej cecha na każdym miejscu ma 1. W takim przypadku nie możemy już zareprezentować następną liczbę bo nie mamy dla tego wolnych bitów. Algorytm polega na tym żeby zwiększać liczbę dopóki ona nie będzie ∞.

1.3.3 Wyniki

W wyniku działania programu otrzymano następujące wyniki dla poszczególnych arytmetyk:

Arytmetyka `Float16`
Wartość uzyskana w wyniku działania algorytmu: $6.55 \cdot 10^4$
Wartość domyślna dla arytmetyki `Float16`: $6.55 \cdot 10^4$
Wartość dla języka `C` w pliku nagłówkowym `float.h`: *nie zdefiniowano*

Arytmetyka `Float32`
Wartość uzyskana w wyniku działania algorytmu: $3.4028235 \cdot 10^{38}$
Wartość domyślna dla arytmetyki `Float32`: $3.4028235 \cdot 10^{38}$
Wartość dla języka `C` w pliku nagłówkowym `float.h`: $3.4028235 \cdot 10^{38}$

Arytmetyka `Float64`
Wartość uzyskana w wyniku działania algorytmu: $1.7976931348623157 \cdot 10^{308}$
Wartość domyślna dla arytmetyki `Float64`: $1.7976931348623157 \cdot 10^{308}$
Wartość dla języka `C` w pliku nagłówkowym `float.h`: $1.7976931348623157 \cdot 10^{308}$

1.3.4 Wnioski

Wartość MAX obliczone przez algorytm są takie same, jak te zwracane przez funkcję `floatmax()`. Wraz ze wzrostem precyzji arytmetyki rośnie również wyliczona wartość MAX. Wynika to z faktu, że im większa precyzja, tym więcej liczb możemy w danej arytmetyce zapisać. Wartości zawarte w pliku nagłówkowym `float.h` są takie same jak dla języka `Julia`.

2 Zadanie II

2.1 Metoda Kahana

Kahan zaproponował metodę polegającą na obliczaniu wartości *macheps* za pomocą następującego wzoru: $3 \cdot (\frac{4}{3} - 1) - 1$.

2.2 Problem

Używając języka `Julia` należało porównać wartości zwracane przez wzór podany przez Kahana z wartościami *macheps* dla zadanych arytmetyk (`Float16`, `Float32`, `Float64`).

2.2.1 Rozwiązanie problemu

Stworzyć funkcje która zgodnie z podanym wzorem liczy *macheps*

2.3 Wyniki

Program wykorzystujący podany wyżej algorytm dał następujące wyniki:

Arytmetyka `Float16`
Wartość uzyskana metodą Kahana: -0.000977
Wartość domyślna dla arytmetyki `Float16`: 0.000977

Arytmetyka `Float32`
Wartość uzyskana metodą Kahana: $1.1920929 \cdot 10^{-7}$
Wartość domyślna dla arytmetyki `Float32`: $1.1920929 \cdot 10^{-7}$

Arytmetyka `Float64`
Wartość uzyskana metodą Kahana: $-2.220446049250313 \cdot 10^{-16}$
Wartość domyślna dla arytmetyki `Float64`: $2.220446049250313 \cdot 10^{-16}$

2.4 Wnioski

Wartości zwrócone przez algorytm są takie same jak faktyczne wartości *macheps* z dokładnością do znaku, tzn. otrzymane wartości *macheps* dla arytmetyk `Float16` i `Float64` mają przeciwny znak (wartość ujemną) niż te zwracane przez funkcję `eps` języka `Julia`. Przy wykorzystaniu tego algorytmu do liczenia *macheps* trzeba mieć świadomość że znak w wyniku może się różnić. Żeby to poprawić możemy zwracać wartość bezwzględną naszego wyniku, tzn. `|wynik|`.

3 Zadanie III

3.1 Obliczanie δ

Zgodnie z założeniami arytmetyki IEEE754, wszystkie reprezentowane liczby rozmieszczone są w pewnych odległościach δ określonych dla poszczególnych przedziałów.

3.2 Problem

Używając języka `Julia` i arytmetyki `Float64` należało sprawdzić, że liczby w przedziale $[1; 2]$ są równomiernie rozmieszczone z krokiem $\delta = 2^{-52}$, czyli każdą liczbę z zadanego przedziału $[1; 2]$ można zapisać jako $x = 1 + kx$, gdzie $k \in \{1, 2, ..., 2^{-52} - 1\}$ oraz $\delta = 2^{-52}$. W następnej kolejności należało sprawdzić rozmieszczenie liczb w przedziałach $[1; \frac{1}{2}]$ oraz $[2; 4]$.

3.2.1 Rozwiązanie problemu

Najpierw tworzę funkcje która w sposób iteracyjny sprawdza krok pomiędzy liczbami. Program działa ale potrzebuje bardzo dużo czasu. Wtedy używając to że każdą liczbę `IEEE 754` możemy zapisać jak ±m·2c gdzie m to mantysa, a c to cecha tworzymy nową funkcję ktorą wylicza za pomocą podanej wcześniej formuły liczbę rzeczewistą z przedziału z taką samą cechą, krokiem pomiędzy każdymi dwiema liczbami z takiego przedziału jest wyliczona liczba.

3.3 Wyniki

Przedział: $[1; 2]$
Wartość δ: 2^{-52}

Przedział: $[\frac{1}{2}; 1]$
Wartość δ: 2^{-53}

Przedział: $[2; 4]$
Wartość δ: 2^{-51}

3.4 Wnioski

Na podstawie uzyskanych wyników należy stwierdzić, że wzór $x = 1 + k\delta$ oraz wartość $\delta = 2^{-52}$ rzeczywiście opisują właściwe rozmieszczenie liczb w przedziale $[1; 2]$. Ponadto, wyznaczone wartości δ dla pozostałych dwóch przedziałów pozwalają wnioskować, iż wraz z oddalaniem się od wartości 0.0 odstępy pomiędzy liczbami są większe. Jednocześnie, w każdym z badanych przedziałów liczb jest dokładnie tyle samo, ponieważ zauważamy, że cecha dla wszystkich liczb z danego przedziału jest taka sama, a zmianie ulega jedynie mantysa.

4 Zadanie IV

4.1 Problem nieodwracalności dzielenia

W zadaniu należało zbadać problem nieodwracalności dzielenia w arytmetyce `Float64` dla języka `Julia`.

4.2 Problem

Zgodnie z poleceniem zadnia, należało eksperymentalnie, stosując arytmetykę `Float64`, obliczyć wyrażenie $fl(x \cdot fl(\frac{1}{x})) \neq 1$, gdzie fl oznacza arytmetykę, w której chcemy wykonać powyższe działanie. Należało znaleźć najmniejszą liczbę spełniającą powyższą zależność.

4.2.1 Rozwiązanie problemu

Zostały stworzone dwie funkcji sprawdzające bląd zaokrąglenia przy dzieleniu. W pierwszej funkcji szukamy najmniejszą taką liczbę większą od jedynki, w drugiej większą od zera.

4.3 Wyniki

Otrzymana wartość dla $(1 < x < 2)$: 1.000000057228997.
Otrzymana wartość dla najmniejszej takiej liczby że $0 < x$: $5.0e - 324$
(Jest to najmniejsza liczba większa od 0 dla arytmetyki `Float64`)

4.4 Wnioski

Fakt, iż dla arytmetyki `Float64` nie zachodzi równość $x \cdot \frac{1}{x} = 1$ wynika z błędów zaokrągleń, które powstają podczas wykonywania działań wykonywanych przy użyciu tej arytmetyki.

5 Zadanie V

5.1 Obliczanie iloczynu skalarnego

W zadaniu 5. należało przeprowadzić eksperyment polegający na obliczeniu iloczynu skalarnego zadanych dwóch wektorów:

$$X = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$
$$Y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$$

5.2 Problem

Używając języka `Julia` należało przeanalizować działanie oraz wyniki zwracane przez cztery różne algorytmy obliczające iloczyn skalarny wektorów X i Y.

5.2.1 Rozwiązanie problemu

Tworzenie funkcji wykonujących każdy z czterech zaproponowanych metod liczenia iloczynu skalarnego.

5.3 Wyniki

W wyniku działania powyższych programów uzyskano następujące wyniki dla poszczególnych rozwiązań:

(a) Arytmetyka `Float32`: -0.3472038161853561
Arytmetyka `Float64`: $1.0251881368296672 \cdot 10^{-10}$

(b) Arytmetyka `Float32`: -0.3472038162872195
Arytmetyka `Float64`: $-1.5643308870494366 \cdot 10^{-10}$

(c) Arytmetyka `Float32`: -0.5
Arytmetyka `Float64`: 0.0

(d) Arytmetyka `Float32`: -0.5
Arytmetyka `Float64`: 0.0

Prawidłowy wynik iloczynu skalarnego dla wektorów X i Y to $-1.00657107000000 \cdot 10^{-11}$ (wynik z dokładnością do 15 cyfr znaczących).

5.4 Wnioski

Wnioskiem z uzyskanych wyników jest fakt, że kolejność wykonywania działań na liczbach zmiennopozycyjnych ma duże znaczenie dla uzyskanego wyniku. Wraz ze wzrostem precyzji arytmetyki rośnie również precyzja uzyskanego wyniku, aczkolwiek nawet dla obliczeń w arytmetyce `Float64` nie udało się uzyskać dokładnej wartości. Można zauważyć również, że na wielkość generowanego błędu wpływa rząd wielkości dodawanych do siebie liczb, np. jeżeli do liczby a dodamy liczbę b mniejszą od a o kilkanaście rzędów wielości, to wtedy wygenerujemy względnie niewielki błąd.

6 Zadanie VI

6.1 Obliczanie wartości funkcji

W zadaniu 6. należało przy użyciu języka `Julia` policzyć w arytmetyce `Float64` wartości dwóch zadanych funkcji rzeczywistych.

6.2 Problem

Zgodnie z poleceniem zadania, używając arytmetyki `Float64`, należało policzyć wartości dla funkcji $f(x) = \sqrt{x^2+1} - 1$ oraz $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ dla wartości $x \in \{8^{-1}, 8^{-2}, 8^{-3}, ...\}$.

6.2.1 Rozwiazanie problemu

Zostały zaimplementowane dwie funkcje do sprawdzenia tego jak zaokrąglenie liczb zaburza wyniki.

6.3 Wyniki

x	$f(x)$	$g(x)$
$x = 8^{-1}$	$f(x) = 0.0077822185373186414$	$g(x) = 0.0077822185373187065$
$x = 8^{-2}$	$f(x) = 0.00012206286282867573$	$g(x) = 0.00012206286282875901$
$x = 8^{-3}$	$f(x) = 1.9073468138230965 \cdot 10^{-6}$	$g(x) = 1.907346813826566 \cdot 10^{-6}$
...	...	...
$x = 8^{-8}$	$f(x) = 1.7763568394002505 \cdot 10^{-15}$	$g(x) = 1.7763568394002489 \cdot 10^{-15}$
$x = 8^{-9}$	$f(x) = 0.0$	$g(x) = 2.7755575615628914e \cdot 10^{-17}$
...	...	...

Podczas działania programu obliczono wartości funkcji f i g dla 10 pierwszych argumentów.

6.4 Wnioski

Funkcje f i g są sobie równe. Dla kilku pierwszych wartości obliczone przez program wartości rzeczywiście są zbliżone, jednak wraz ze zmniejszaniem się wartości argumentu x widzimy co raz większe rozbieżności w wynikach. Funkcja f już dla argumentu $x = 8^{-9}$ osiągnęła wartość 0.0. Funkcja g osiągnęła wartość 0.0 dopiero dla argumentu $x = 8^{-179}$.W rzeczewistośći funkcja f i g "zbliżają się" do wartości 0 dla kolejnych argumentów, ale nigdy tej wartości nie osiągną. Otrzymane wyniki stoją w sprzeczności z tym faktem, jednak nie są one sprzeczne z zasadą działania arytmetyki `Float64`. Zauważmy, że w przypadku funkcji f wartość pierwiastka jest bardzo zbliżona do wartości 1.0, zatem odejmowane są od siebie bardzo bliskie liczby w wyniku czego, dochodzi do utraty cyfr znaczących. W przypadku funkcji g nie jest wykonywane żadne odejmowanie przez co dokładność wyników jest większa a wszelkie uzyskane błędy wynikają z precyzji arytmetyki.

7 Zadanie VII

7.1 Obliczanie pochodnej funkcji

W zadaniu 7. należało, używając języka `Julia`, policzyć pochodną funkcji w arytmetyce `Float64` za pomocą zadanego wzoru, a następnie obliczyć błąd względny uzyskanych przybliżeń.

7.2 Problem

Przybliżoną wartość pochodnej funkcji f można obliczyć za pomocą następującego wzoru:

$$f^{'}(x) \approx \tilde{f}^{'}(x) = \frac{f(x_0+h)-f(x_0)}{h}$$

Obliczenia należało wykonać w arytmetyce `Float64`. Dana była funkcja $f(x) = \sin(x) + \cos(3x)$ oraz punkt $x_0 = 1$, w którym liczona była pochodna funkcji f. Należało również obliczyć błąd zadany wzorem $\mid f^{'}(x) - \tilde{f}^{'}(x) \mid$ dla $h = 2^{-n}$, gdzie $n \in \{0, 1, 2, ..., 54\}$.

7.2.1 Rozwiązanie problemu

Tworzymy dwie funkcje obliczające pochodne funkcji $f(x) = \sin(x) + \cos(3x)$ za pomocą wzoru przybliżonej pochodnej i drugą która już ma przekształconą funkcje na funkcje pochodnej. Następnie sprawdzamy błąd odejmując wartości otrzymane. Iterujemy się po n zmieniając h

7.3 Wyniki

h	$\tilde{f'}(x)$	$\mid f'(x) - \tilde{f'}(x) \mid$	$1 + h$
2^{0}	2.0179892252685967	2.1665107370611456	2.0
2^{-1}	1.8704413979316472	2.1665107370611456	1.5
2^{-2}	1.1077870952342974	1.2563086070268463	1.25
2^{-3}	0.6232412792975817	0.7717627910901306	1.125
...	...	...	...
2^{-15}	0.11706539714577957	0.2655869089383285	1.000030517578125
2^{-16}	0.11700383928837255	0.26552535108092146	1.0000152587890625
2^{-17}	0.11697306045971345	0.26549457225226236	1.0000076293945312
...	...	...	...
2^{-27}	0.11694231629371643	0.26546382808626534	1.0000000074505806
2^{-28}	0.11694228649139404	0.26546379828394295	1.0000000037252903
2^{-29}	0.11694222688674927	0.2654637386792982	1.0000000018626451
2^{-30}	0.11694216728210449	0.2654636790746534	1.0000000009313226
...	...	...	...
2^{-36}	0.116943359375	0.2654648711675489	1.000000000014552
2^{-37}	0.1169281005859375	0.2654496123784864	1.000000000007276
2^{-38}	0.116943359375	0.2654648711675489	1.000000000003638
...	...	...	
2^{-51}	0.0	0.1485215117925489	1.0000000000000004
2^{-52}	-0.5	0.3514784882074511	1.0000000000000002
2^{-53}	0.0	0.1485215117925489	1.0
2^{-54}	0.0	0.1485215117925489	1.0

7.4 Wnioski

Wraz z zmniejszaniem się wartości h zauważamy, że wartości $1 + h$ są coraz mniej dokładne, a dla najmniejszych h są równe 1.0. Wnioskiem płynącym z obserwacji jest fakt, iż należy unikać odejmowania od siebie wartości znacząco różniących się swoim wykładnikiem, ponieważ wpływa to na dokładność otrzymanych wyników. Na zaburzenie dokładności wyniku (utratę cyfr znaczących) wpływa również odejmowanie od siebie bardzo bliskich sobie wartości (dla bardzo małych h).