Politechnika Wrocławska

Anna Modrzejewska

236642

Obliczenia naukowe

Lista nr 1

19 października 2018

Zadanie 1

1) Opis problemu

Należy iteracyjnie wyznaczyć wartość epsilonu maszynowego *macheps*, liczbę *eta* oraz liczbę MAX w arytmetyce Float16, Float32 oraz Float64.

2) Rozwiązanie

Do wyznaczenia wartości *macheps* można posłużyć się pętlą:

```
x = fl(1.0)
while fl(1.0+x/2)>1.0
   x=fl(x/2)
end
```

Do wyznaczenia liczby *eta*:

```
x = fl(1.0)
while fl(x/2)>0.0
   x=fl(x/2)
end
```

Do wyznaczenia liczby MAX (funkcja $isinf(x)$ zwraca true w przypadku, gdy argument jest nieskończony, false w przeciwnym):

```
x = fl(1.0)
while !isinf(fl(x*2))
   x=fl(x*2)
end
x=x*(2-eps(fl))
```

3) Otrzymane wyniki

	macheps	eta	max
Float16	0.000977	$6.0e^{-8}$	$6.55e^{4}$
Float32	$1.1920929e^{-7}$	$1.0e^{-45}$	$3.4028235e^{38}$
Float64	$2.220446049250313e^{-16}$	$5.0e^{-324}$	$1.7976931348623157e^{308}$

4) Analiza wyników

Otrzymane przez program wyniki są takie same, co wartości zwracane przez funkcje $eps()$, $nextfloat(fl(0.0))$, $realmax()$ oraz FLT_EPSILON, DBL_EPSILON, FLT_MAX, DBL_MAX z pliku nagłówkowego float.h języka C.

Zgodnie z definicją podaną na wykładzie, precyzją arytmetyki nazywa się największy błąd względny dla przedstawienia liczby w arytmetyce zmiennopozycyjnej (zaokrąglenia). Zatem wartość epsilonu maszynowego (która została wyżej wyznaczona jako najmniejsza taka liczba, która spełnia warunek $x + 1.0 > 1.0$) równa się precyzji arytmetyki.

Zadanie 2

1) Opis problemu

Należy sprawdzić słuszność wzoru Kahana, według którego epsilon maszynowy można otrzymać obliczając wartość wyrażenia: $3 * (\frac{4}{3} - 1) - 1$.

2) Otrzymane wyniki

	ze wzoru Kahana	z funkcji eps()
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

3) Analiza wyników i wnioski

Można zauważyć, że wyniki z wzoru Kahana są równe z epsilonem maszynowym co do modułu, zatem teza Kahana jest słuszna. Jednak otrzymane wartości różnią się od 0, czego powodem może być to, że wartość ułamka $\frac{4}{3}$ w reprezentacji liczb zmiennoprzecinkowych nie jest dokładną wartością, tylko liczbą zbliżoną do $\frac{4}{3}$.

Zadanie 3

1) Opis problemu

Należy sprawdzić, że w arytmetyce Float64 liczby z zakresu $[1, 2]$ są rozmieszczone równomiernie z krokiem $\delta = 2^{-52}$ oraz sprawdzić, jak rozmieszczone są w przedziałach $[\frac{1}{2}, 1]$ i $[2, 4]$

2) Rozwiązanie

Program przy pomocy funkcji *bits* sprawdza zapis bitowy liczby 1.0, następnej po 1.0 (przy użyciu funkcji nextfloat) oraz liczby $1 + 2^{-52}$, a także zapis bitowy liczby $2 - 2^{-52}$, następnej po niej i liczby 2.0.

3) Otrzymane wyniki i analiza

```
julia> bits(1.0)
0011111111110000000000000000000000000000000000000000000000000000
julia> bits(nextfloat(1.0)
0011111111110000000000000000000000000000000000000000000000000001
julia> bits(1.0+2.0^(-52))
0011111111110000000000000000000000000000000000000000000000000001
```

Można wywnioskować, że kolejna po 1.0 liczba jest oddalona od niej o $\delta = 2^{-52}$. Podobnie z liczbą 2.0:

```
julia> bits(2.0)
0100000000000000000000000000000000000000000000000000000000000000
julia> bits(2.0-2.0^(-52))
```

```
0011111111111111111111111111111111111111111111111111111111111111
julia> bits(nextfloat(2.0-2.0^(-52))
0100000000000000000000000000000000000000000000000000000000000000
```

Widać, że po poprzednia liczba od 2 jest oddalona od niej o dokładnie deltę. Dla przedziału $[\frac{1}{2}, 1]$ krokiem okazała się $\delta = 2^{-53}$, natomiast dla $[2, 4]$ $\delta = 2^{-51}$:

```
julia> bits(0.5)
0011111111100000000000000000000000000000000000000000000000000000
julia> bits(0.5+2.0^(-53))
0011111111100000000000000000000000000000000000000000000000000001
julia> bits(2.0)
0100000000000000000000000000000000000000000000000000000000000000
julia> bits(2.0+2.0^(-51))
0100000000000000000000000000000000000000000000000000000000000001
```

Można zauważyć, że dla przedziału $[2^m, 2^{m+1}]$ im większe m, tym większa delta, czyli mniejsze zagęszczenie liczb. Zmiana delty następuje w momencie zmiany cechy liczby (bity od 2. do 12.). Dla liczb z przedziału $[\frac{1}{2}, 1]$ cechą jest 01111111110, dla $[1, 2]$ jest to 01111111111, dla $[2, 4]$: 10000000000. Wzrost cechy o 1 powoduje dwukrotny wzrost delty.

Zadanie 4

1) Opis problemu

Należy znaleźć w arytmetyce Float64 takie najmniejsze $x \in (1, 2)$, że $x * \frac{1}{x} \neq 1$

2) Rozwiązanie

Program liczy wartość wyrażenia $x * \frac{1}{x}$ w pętli, biorąc za x kolejne wartości (za pomocą funkcji nextfloat()) do momentu, aż będzie się różniło od 1.

3) Wynik

Znaleziony x ma wartość $x = 1.000000057228997$. Wartość podanego wyrażenia to 0.9999999999999999, a jego reprezentacja bitowa to 0011111111101111111111111111111111111111111111111111111111111111

Zadanie 5

1) Opis problemu

Należy zaimplementować 4 algorytmy obliczające iloczyn skalarny dwóch wektorów w arytmetyce Float32 oraz Float64. Algorytmy różnią się kolejnością sumowania składników.

2) Rozwiązanie

Algorytm A liczy sumę składników "wprzód", czyli dodaje od początku do końca. Algorytm B liczy sumę od końca do początku. Algorytm C tworzy nowy wektor $z = [x_1y_1, x_2y_2, x_3y_3, x_4y_4, x_5y_5$, sortuje rosnąco, a następnie liczby dodatnie sumuje od największej do najmniejszej, a ujemne od najmniejszej do największej. Algorytm D liczy sumę odwrotnie do algorytmu C.

3) Wyniki

	Algorytm A	Algorytm B	Algorytm C	Algorytm D
Float32	-0.4999443	-0.4543457	-0.5	-0.5
błąd	0.49994429944939167	0.4543457031149	0.4999999999899	0.4999999999899
Float64	$1.02518813683 * 10^{-10}$	$-1.5643308870494 * 10^{-10}$	0.0	0.0
błąd	$1.1258452438296672 * 10^{-10}$	$1.4636737800494365 * 10^{-10}$	$1.00657107 * 10^{-11}$	$1.00657107 * 10^{-11}$

Rzeczywista wartość iloczynu skalarnego: $-1.00657107000000 * 10^{-11}$

4) Analiza wyników

Najbliższe rzeczywistego wyniku są rezultaty algorytmu C i D liczące w arytmetyce Float64. Na podstawie różności wyników można wywnioskować, że kolejność sumowania ma znaczenie. Najlepiej sumować ze sobą w pierwszej kolejności najbliższe sobie rzędem wielkości, a na końcu najdalsze.

Zadanie 6

1) Opis problemu

Należy policzyć w arytmetyce podwójnej precyzji wartości funkcji $f(x) = \sqrt{x^2 + 1} - 1$ oraz $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ dla $x = 8^{-1}, 8^{-2}$....

2) Otrzymane wyniki

Po wywołaniu funkcji $f(x)$ oraz $g(x)$ z powyższymi argumentami wyniki będą następujące:

```
julia> f(8.0^-1)
0.0077822185373186414

julia> g(8.0^-1)
0.0077822185373187065
```

```
julia> f(8.0^-2)
0.00012206286282867573

julia> g(8.0^-2)
0.00012206286282875901
```

```
julia> f(8.0^-3)
1.9073468138230965e-6

julia> g(8.0^-3)
1.907346813826566e-6
```

```
julia> f(8.0^-8)
1.7763568394002505e-15

julia> g(8.0^-8)
1.7763568394002489e-15
```

```
julia> f(8.0^-9)
0.0

julia> g(8.0^-9)
2.7755575615628914e-17
```

```
julia> f(8.0^-10)
0.0

julia> g(8.0^-10)
4.336808689942018e-19
```

3) Analiza wyników i wnioski

Można zauważyć, że mimo $f = g$, wyniki dla tych samych argumentów nieznacznie różnią się, szczególnie dla $x \leqslant 8^{-9}$, dla którego funkcja $f(x)$ zwraca 0.0. Wynika to prawdopodobnie z tego, że w przypadku f(x), odejmując 1 od liczby zbliżonej do 1 następuje redukcja znaczących cyfr.

Zadanie 7

1) Opis problemu

Należy policzyć w arytmetyce Float64 przybliżoną wartość pochodnej funkcji $f(x) = sinx + cos3x$ w punkcie $x_0 = 1$ oraz błąd bezwzględny.

2) Rozwiązanie

Program składa się z funkcji *df* wyliczającej pochodną z ogólnego wzoru $f'(x_0) = \frac{f(x_0+h)-f(x_0)}{h}$, funkcji $f(x) = sin(x) + cos(3x)$ oraz funkcji $df2$ liczącej rzeczywistą pochodną funkcji $f(x)$: $f'(x) = cos(x) - 3sin(3x)$ w celu porównania wyników funkcji *df* i $df2$.

3) Otrzymane wyniki

Funkcja *df* została przetestowana dla $h = 2^{-n}$, gdzie $n \in \{0, 1, 2, ..., 54\}$:

h	wynik df	błąd
2^{0}	2.0179892252685967	1.9010469435800585
2^{-1}	1.8704413979316472	1.753499116243109
2^{-2}	1.1077870952342974	0.9908448135457593
...	...	...
2^{-20}	0.11694612901192158	3.8473233834324105e-6
2^{-21}	0.1169442052487284	1.9235601902423127e-6
2^{-22}	0.11694324295967817	9.612711400208696e-7
...	...	...
2^{-47}	0.109375	0.007567281688538152
2^{-48}	0.09375	0.023192281688538152
2^{-49}	0.125	0.008057718311461848
2^{-50}	0.0	0.11694228168853815
2^{-51}	0.0	0.11694228168853815
2^{-52}	-0.5	0.6169422816885382
2^{-53}	0.0	0.11694228168853815
2^{-54}	0.0	0.11694228168853815

4) Analiza wyników i wnioski

Można zauważyć, że dla $n \leqslant -50$ funkcja zaczyna zwracać tę samą wartość 0.0, ponieważ odejmujemy wartość $f(x_0)$ od liczby do niej zbliżonej (podobnie, jak w zadaniu wyżej - następuje redukcja znaczących cyfr), dlatego coraz mniejsze h będzie generowało taki sam błąd. Wynika to z niewystarczającej precyzji arytmetyki.