Wiktor Satora 411502 gr.2

Język C++, Kompilator MinGW

System Windows 10 x64, Program CLion

MOWNiT - Sprawozdanie lab1

Polecenie:

2. Wykonać obliczenia (dla zmiennych typu *float, double, long double*) wg podanych poniżej wzorów dla 101 równoodległych wartości x z przedziału [0.99, 1.01]:

- $f(x) = x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1$
- $f(x) = (((((((x - 8)x + 28)x - 56)x + 70)x - 56)x + 28)x - 8)x + 1$
- $f(x) = (x - 1)^8$
- $f(x) = e^{(8\ln(\mathrm{abs}(x-1)))}$, $x \neq 1$

Porównać wyniki. Objaśnić różnice w wynikach.

Wykonanie:

Funkcje przygotowujące tablice z odpowiednimi danymi:

Uzupełniają tablice danymi odpowiednimi typami

```cpp
float* prepare_floats(){
    static float floats[101];
    float left=0.99;
    float right=1.01;
    float span=(right-left)/100;
    for (int i = 0; i < 101; ++i) {
        floats[i]=left+i*span;
    }
    return floats;
}
```

```cpp
double* prepare_doubles(){
    static double doubles[101];
    double left2=0.99;
    double right2=1.01;
    double span2=(right2-left2)/100;
    for (int i = 0; i < 101; ++i) {
        doubles[i]=left2+i*span2;
    }
    return doubles;
}
```

```cpp
long double* prepare_long_doubles(){
    static long double long_doubles[101];
    long double left3=0.99;
    long double right3=1.01;
    long double span3=(right3-left3)/100;
    for (int i = 0; i < 101; ++i) {
        long_doubles[i]=left3+i*span3;
    }
    return long_doubles;
}
```

Techniczne aspekty
Funkcje potęgujące wbudowane: powf-potęguje floaty, pow-potęguje double'y, powl-potęguje long_double'y

Funkcje abs (wartość bezwzględna) log (logarytm naturalny) oraz exp (funkcja eksponencjalna) dostosowywują typ do przypisywanej zmiennej

Krótka charakteryzacja typów danych:
- Float - 32-bitowa liczba zmiennoprzecinkowa, z czego 1 bit służy do reprezentacji znaku, 8 bitów na cechę (wykładnik) i 23 bity na mantysę (część ułamkowa). Oznacza to, że liczby zmiennoprzecinkowe typu float są reprezentowane z około 6-7 cyframi dokładności.
- Double - o 64-bitowa liczba zmiennoprzecinkowa, z czego 1 bit służy do reprezentacji znaku, 11 bitów na cechę (eksponent) i 52 bity na mantysę (część ułamkowa). Oznacza to, że liczby zmiennoprzecinkowe typu double są reprezentowane z około 15-16 cyframi dokładności.
- Long double - precyzja i zakres zależą od implementacji, jednak zazwyczaj jest to 80 lub 128-bitowa liczba zmiennoprzecinkowa. Oznacza to, że liczby zmiennoprzecinkowe typu long double są reprezentowane z około 19-20 cyframi dokładności lub nawet więcej

Oszacowanie zmienności wyników:
Przedział [0.99,1.01] dzielimy na 100, więc skok pomiędzy nimi będzie wynosił 0.0002 (2*10^-4), a według polecenia podnosimy wszystko do 8 potęgi więc:

$$(0.0002)^8 = 2^8 * 10^{(-4 * 8)} = 256*10^{(-32)}$$

Zatem wyniki mogą różnić się na 32 miejscu po przecinku

Funkcje z polecenia
Warto zauważyć, że wszystkie cztery funkcje podane w poleceniu są tymi samymi po odpowiednich przekształceniach. Czy zatem będą dawały takie same wyniki?

Porównanie wyników dla różnych typów danych
Wykonując program otrzymujemy masę wyników, tu jeden z nich, przykładowy:

Przy obliczeniu wartości funkcji dla wartości **1.0098**, czyli **(0.0098)^8** (wzór trzeci) takie wartości prezentują się kolejno dla typu float, double oraz long double:

0.0000000000000000850733356268168559432413

0.0000000000000000850763022581806987698841

0.0000000000000000850763022581791630032538

Tutaj w wersji naukowej:

8.507333562681685594324132448917907822761 e-17

8.507630225818069876988406721600742531312 e-17

8.507630225817916300325375745308763827231 e-17

Wniosek: widać, że początkowe cyfry niezerowe pokrywają się, później tracą dokładność.
Pierwszy trafi **float** (stracił dokładność po 4 cyfrach),
następnie stracił **double** (po 12 cyfrach), niestety **long double'a** ciężko oszacować kiedy traci dokładność, lecz bez wątpienia jest on najdokładniejszy.

Wniosek 2: dodatkowo widać z notacji naukowej, że sposób pamiętania wartości w kolejnych typach danych jest następujący:

Ilość cyfr którą może poprawnie zapamiętywać dany typ jest liczona od pierwszej niezerowej, przesunięciem zajmuje się cecha.

Porównanie przez odejmowanie

- od long double'a floata:

0.0000000000000000000029666313623070600124

- od long double'a double'a:

0.0000000000000000000000000000015357666303

widać różniące się końcówki jeszcze bardziej.

Teraz to samo, ale dla wzoru 4:

0.0000000000000000850733885663760593370125

0.0000000000000000850763022581806494660775

0.0000000000000000850763022581791629129758

Notacja naukowa:

8.507338856637605933701251625933537070523 e-17

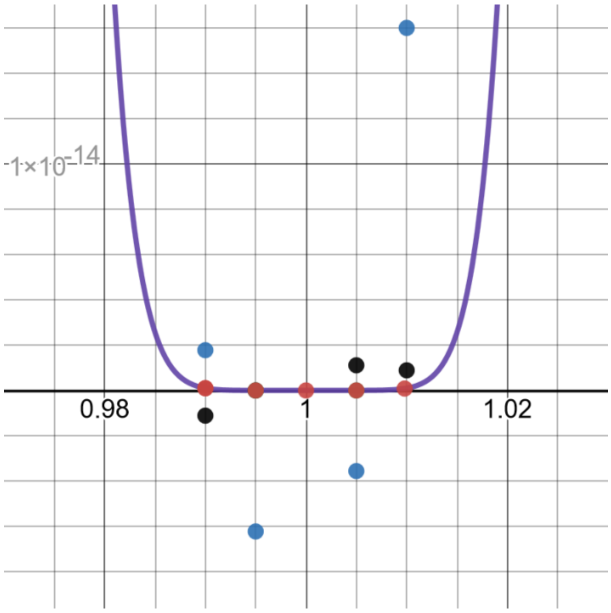
8.507630225818064946607749090276958708008 e-17

8.507630225817916291297579130993595766031 e-17

Wniosek 3: Tutaj sprawa się ma identycznie, aczkolwiek widzimy, że końcówki liczb są inne. Dzieje się tak dlatego, że kolejność liczenia konkretnych działań w funkcji ma znaczenie. Wynika to z niedokładności reprezentacji liczb zmiennoprzecinkowych w pamięci komputera

Wykres

Na wykresie przedstawiona jest właściwa funkcja (na fioletowo). Punkty to są poszczególne wyniki dla typu *double*, dla kolejnych wzorów (1-niebieski, 2-czarny, 3-zielony, 4-czerwony). Zielone punkty pokrywają się niemalże dokładnie z czerwonymi, dlatego są niewidoczne na wykresie.



Funkcje 1 i 2 mają duże rozbieżności, czasami nawet wypadają w wartości ujemne.
Natomiast funkcje 3 i 4 prawie idealnie odzwierciedlają właściwą funkcję, co może zaskakiwać. Dzieje się tak przez szereg działań dodawania i mnożenia w funkcjach 1 i 2.

Wniosek 4: Funkcje 3 i 4 pomimo potęgowania, liczenia wartości bezwzględnej oraz logarytmu naturalnego dużo dokładniej odzwierciedlają właściwą funkcję niż **funkcje 1 i 2**, które znacznie tracą dokładność przez szereg dodawań i mnożeń.