Sprawozdanie 1

Piotr Kotara

Marzec 2019

1 Napisz program, który wyznaczy epsilon maszynowe dla typu float i double w języku C oraz float w Python przy pomocy programu rekurencyjnego.

Kod w C++:

```
#include <iostream>

using namespace std;


float epsilon(float curr){
        if(1.0f + curr/2 != 1.0f){
                return epsilon(curr/2);
        }
        else return curr;
}


double epsilon(double curr){
        if(1.0 + curr/2 != 1.0){
                return epsilon(curr/2);
        }
        else return curr;
}

int main(){
        cout << "epsilon dla floata: " << epsilon(1.0f) << endl;
        cout << "epsilon dla double'a: " << epsilon(1.0) << endl;

}
```

Wyniki programu:

```
epsilon dla floata: 1.19209e-07
epsilon dla double: 2.22045e-16
```

Wartości zgadzają się z wartościami na wikipedii. Zaobserwowano, że w zależności od tego czy dodajemy czy odejmujemy od jedynki epsilon różni sie o rząd wielkości w systemie dwójkowym.

Kod w Pythonie:

```python
def epsilon(curr:float):
        if(1.0 + curr/2 != 1.0):
                return epsilon(curr/2)
        else:
        return curr

print("Epsilon floata: ", epsilon(1.0))
```

Wyniki programu:

```
Epsilon floata: 2.220446049250313e-16
```

Wynik jest tożsamy z wynikiem dla double'a w C++, co może wskazywać, że float w Pythonie to liczba zmiennoprzecinkowa o podwójnej precyzji. Po sprawdzeniu w dokumentacji Pythona okazyje się, że rzecywiście tak jest.

2 Napisz dwa programy w języku C bądź Python, gdzie pierwszy zachowuje się niestabilnie i wyjaśnij dlaczego, podczas gdy drugi zachowuje się stabilnie i jest ulepszoną wersją pierwszego programu.

Jako przykład wybrano algorytm obliczania wartości funkcji e^x z jej rozwinięcia w szereg Taylora:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Pierwszym naiwnym sposobem jest próba liczenia sumy po prostu poprzez sumowanie wyrazów w pętli:

```
def exp(x:int, acc: int):
    res = 0
    for i in range (0, acc):
            res += (x**i)/fact(i)

return res
```

Rozwiązanie to prowadzi do dużych rozbieżności dla $x < 0$. Dzieje się tak, gdyż zadzodzi zjawisko zwane *Catastrophic cancellation*. Polega ono na utracie bitów przy odejmowaniu bardzo bliskich sobie liczb.

Rozwiązaniem problemu jest skorzystanie z faktu, że $e^{-x} = \frac{1}{e^x}$. Oto poprawiony algorytm:

```
def bttrexp(x:int, acc: int):
res = 0
if x < 0: return (1/bttrexp(-x, acc))
for i in range (0, acc):
        res += (x**i)/fact(i)

return res
```

Oto cały kod programu:

```
import math

def fact(x :int):
        return 1 if x == 0 else fact(x-1)*x

def exp(x:int, acc: int):
        res = 0
        for i in range (0, acc):
                res += (x**i)/fact(i)

        return res

def bttrexp(x:int, acc: int):
        res = 0
        if x < 0: return (1/bttrexp(-x, acc))
        for i in range (0, acc):
                res += (x**i)/fact(i)

        return res

print("Gorsze: ", exp(-30, 300), "Lepsze: ", bttrexp(-30, 300), "Z biblioteki math: ", math.exp(-30))
```

Dla zadanych wartości otrzymujemy wynik

```
Gorsze: -8.553016433669241e-05 Lepsze: 9.357622968840171e-14 Z biblioteki math: 9.357622968840175e-14
```

Jak widać wynik naiwnego algorytmu odbiega od rzeczywistości w przeciwieństwie do poprawionej wersji.

3 Sumowanie liczb pojedynczej precyzji w języku C:

3.1 Napisz program, który oblicza sumę N liczb pojedynczej precyzji przechowywanych w tablicy o N= 10^7 elementach. Tablica wypełniona jest tą samą wartością v z przedziału [0.1,0.9] np.v= 0.53125. Zaproponuj dwa takie v, gdzie błędy będą najmniejsze i największe w czasie sumowania.

- Napisz program, który oblicza sumę N liczb pojedynczej precyzji przechowywanych w tablicy o $N = 10^7$ elementach. Tablica wypełniona jest tą samą wartością v z przedziału [0.1,0.9] np.v= 0.53125. Zaproponuj dwa takie v, gdzie błędy będą najmniejsze i największe w czasie sumowania.
- Wyznacz bezwzględny i względny błąd obliczeń. Dlaczego błąd względny jest tak duży?
- W jaki sposób rośnie błąd względny w trakcie sumowania? Przedstaw wykres (raportuj wartość błędu co 25000 kroków) i dokonaj jego interpretacji.

Kod programu:

```
#include <iostream>
#include <iomanip>
#include <ctime>
```

```
#define e7 10000000
#define C 0.3728
using namespace std;

float normal_sum();


float *tab;
int main(){
        cout << setprecision(12);
        clock_t start, end;

        tab = new float[e7];
        for(int i = 0; i < e7; i++) tab[i] = C;

        start = clock();
        float sum = normal_sum();
        end = clock();
        cout << "\nNormal adding time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========";
        cout << "\nSUM/ERROR/RERROR  " <<sum<<" "<< (e7*C - sum) <<" " << (e7*C - sum)/(C*e7) <<  end


}

float normal_sum(){

        float sum = 0;

        for(int i = 0; i < e7; i++){
                sum += tab[i];
                if(i % 25000 == 0){

                // cout << i<< ": " << (i*C - sum) << endl;
                cout <<sum<<" "<< (i*C - sum) <<" " << (i*C - sum)/(C*e7) <<  endl;
                }
        }
        cout << "#DATA_END\n";
        return sum;
}
```

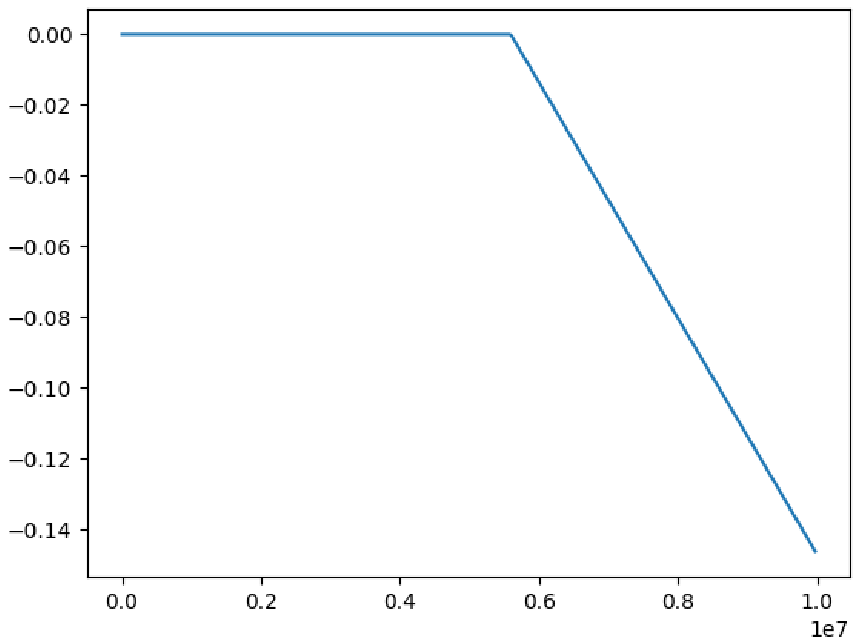
Wynik jest dokładny dla $v = 0.25$.

```
Normal adding time elapsed: 0.035898 ===========
SUM/ERROR/RERROR 2500000 0 0
```

Wynik jest silnie zaburzony dla $v = (0.25 + 0.125)/2$ (Dobrany tak żeby był daleko od potęg 2).

```
Normal adding time elapsed: 0.055726 ===========
SUM/ERROR/RERROR 2150474.5 -275474.5 -0.146919733333
```

Dla tej wartości wyplotowano wykres błędu względnego od iteracji sumy skryptem napisanym w Pythonie: Jak widzimy



Rysunek 1: Wykres błędu względnego od iteracji sumy dla v = (0.25+0.125)/2

błąd na początku jest bliski zeru a następnie rośnie na moduł liniowo. Najprawdopodobniej jest to spowodowane, że od pewnej iteracji suma jest na tyle duża w porównaniu do składnika, że przy dodawaniu ucinane są mało znaczące bity po tym jak wyrównane zostaną wykładniki.

- Zaimplementuj rekurencyjny algorytm sumowania.
- Wyznacz bezwzględny i względny błąd obliczeń. Dlaczego błąd względny znacznie zmalał?
- Porównaj czas działania obu algorytmów dla tych samych danych wejściowych.

Kod programu:

```
float recsum(float* tab, int start, int end){
        if(start == end) return tab[start];
```

```
        int mid = start + (end-start)/2;
        return recsum(tab, start, mid) + recsum(tab, mid+1, end);
    }
```

Dla danych wejściowych jak wyżej otrzymujemy:

```
Normal adding time elapsed: 0.035277 ===========
SUM/ERROR/RERROR 2150474.5 -275474.5 -0.146919733333

Recursive adding time elapsed: 0.088499 ===========
SUM/ERROR/RERROR 1875000 0 0
```

Jak widzimy dodawanie rekurencyje zwraca dokładny wynik kosztem ponad 2x dłuższego czasu wykonania. Sumując rekurencyjnie zawsze dodajemy liczby porównywalnych rzędów wielkości co rozwiązuje wcześniejszy problem.

- Przedstaw przykładowe dane wejściowe, dla których algorytm sumowania rekurencyjnego zwraca niezerowy błąd.

Dla v = 0.503940 program zwraca:

```
Recursive adding time elapsed: 0.089134 ===========
SUM/ERROR/RERROR 5039399.5 0.500000000931 9.9218161077e-08
```

---

4 Algorytm Kahana w języku C:

Kod:

```
    float kahanSum(float* tab, int len) {
        float sum = tab[0];
        float compensate = 0.0;
        float tmp, buf;
        for(int i = 1; i < len; i++){
            tmp = tab[i] - compensate;
            buf = sum + tmp;
            compensate = (buf - sum) - tmp;
            sum = buf;
        }
        return sum;
    }
```

Działanie programu opiera się na zachowywaniu młodszych bitów przy sumowaniu i dodawaniu ich do składnika sumy przy następnym dodawaniu.

Dla v = 0.503940:

```
Normal adding time elapsed: 0.035688 ===========
SUM/ERROR/RERROR 5001516.5 37883.5 0.00751746239632

Recursive adding time elapsed: 0.089134 ===========
SUM/ERROR/RERROR 5039399.5 0.500000000931 9.9218161077e-08

Kahan adding time elapsed: 0.104306 ===========
SUM/ERROR/RERROR 5039400 9.31322574615e-10 1.84808226101e-16
```

Cały kod używany w zadaniu 3 i 4:

```
#include <iostream>
#include <iomanip>
#include <ctime>
#define e7 10000000
//#define C (0.25+0.125)/2
#define C 0.503940

using namespace std;

float recsum(float* tab, int start, int end);
float normal_sum();
float kahanSum(float* tab, int len) {
        float sum = tab[0];
        float compensate = 0.0;
        float tmp, buf;
        for(int i = 1; i < len; i++){
                tmp = tab[i] - compensate;
                buf = sum + tmp;
                compensate = (buf - sum) - tmp;
                sum = buf;
        }
        return sum;
}


float *tab;
int main(){
        cout << setprecision(12);
        clock_t start, end;

        tab = new float[e7];
        for(int i = 0; i < e7; i++) tab[i] = C;

        start = clock();
        float sum = normal_sum();
        end = clock();
        cout << "\nNormal adding time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========";
        cout << "\nSUM/ERROR/RERROR  " <<sum<<" "<< (e7*C - sum) <<" " << (e7*C - sum)/(C*e7) <<  endl;


        start = clock();
        sum = recsum(tab,0, e7-1);
        end = clock();
        cout << "\nRecursive adding time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========";
        cout << "\nSUM/ERROR/RERROR  " <<sum<<" "<< (e7*C - sum) <<" " << (e7*C - sum)/(C*e7) <<  endl;


        start = clock();
        sum = kahanSum(tab, e7);
        end = clock();
```

```
        cout << "\nKahan adding time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========";
        cout << "\nSUM/ERROR/RERROR  " <<sum<<" "<< (e7*C - sum) <<" " << (e7*C - sum)/(C*e7) <<  endl;

}

float normal_sum(){

        float sum = 0;

        for(int i = 0; i < e7; i++){
                sum += tab[i];
                if(i % 25000 == 0){

                // cout << i<< ": " << (i*C - sum) << endl;
                cout <<sum<<" "<< (i*C - sum) <<" " << (i*C - sum)/(C*e7) <<  endl;
                }
        }
        cout << "#DATA_END\n";
        return sum;
}

float recsum(float* tab, int start, int end){
        if(start == end) return tab[start];
        int mid = start + (end-start)/2;
        return recsum(tab, start, mid) + recsum(tab, mid+1, end);
}
```

5 Napisz program w języku C, który wyznaczy K najmniejszych liczb ze zbioru N elementowej nieposortowanej tablicy liczb typu float bazując na idei sortowania kubełkowego i algorytmie zaprezentowanym i wytłumaczonym na zajęciach na tablicy. Dokonaj analizy poprawności działania algorytmu i czasu wykonania dla innego algorytmu wyszukującego k K najmniejszych liczb dla zbioru liczb wygenerowanych w zakresach: $< 0.0, 0.3 >$ i $< 0.0, 3.0 >$.

Kod programu:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <list>
#include <ctime>

#define MAXVAL 1.0f
#define MINVAL 0.0f

using namespace std;



float max(const list<float> tab){
        float buff = tab.front();
        for(auto i = tab.begin(); i != tab.end(); ++i){
                if(*i > buff) buff = *i;
        }
        return buff;
}


float min(const list<float> tab){
        float buff = tab.front();
        for(auto i = tab.begin(); i != tab.end(); ++i){
                if(*i < buff) buff = *i;
        }
        return buff;
}

float max(float a, float b){
        return a>b ? a : b;
}

list<float> k_mins( const list<float> workingTab, int k, int n_buck){
        list<float> result;
        if(k == 0) return result;
        //cout << 'D' << flush;
```

```
        float minim = min(workingTab);
        float maxim = max(workingTab);

        float min1 = 0;
        float max1= maxim - minim;

        vector<list<float>> buckets;
        buckets.resize(n_buck);


        for(auto tabElem = workingTab.begin(); tabElem != workingTab.end(); ++tabElem){
                buckets[floor(
                        min((float)(n_buck-1), //min dlatego żeby
                        //maksymalna wartość nie trafiła do kubełka o numerze n_buck
                                max(n_buck+log2(*tabElem-minim), //wszystkie
                                //watrości mniejsze niż 2^(n_buck) do zerowego
                                0.0f)))].push_back(*tabElem);
        }


        //cout << "=========== POTRZEBUJE JESZCZE " << k <<" liczb ================\n";
        for(int i = 0; i < buckets.size(); i++){
                //   cout << "W kubelku nr " << i << " jest " << buckets[i].size() << "liczb" << endl;
                if(result.size() + buckets[i].size() <= k){
                        result.merge(buckets[i]);
                }
                else{
                        result.merge(k_mins(buckets[i], k - result.size(), n_buck));
                        break;
                }
        }
        return result;
}

list<float> classic_k_mins(list<float> workingTab, int k){
        workingTab.sort();
        list<float> res;
        int it = 0;
        for(auto i = workingTab.begin(); i != workingTab.end() && it < k; ++i){
                res.push_back(*i);
                it++;
        }
        return res;
}


int main(){
        srand(time(NULL));
        list<float> test = {2.5, 34, 5.3, 34.5, .43, 5.4, 2, 123094, 542, 0.0432423, 0.0432434};
        list<float> test2;
        const float LIM = 0.3;
        const int K = 10;
```

```
        clock_t start, end;
        list<float> res;

        for(int N = 100; N <= 10000000; N*=10){
                test2.clear();
                cout << "\n\nTesting 10 min of " << N << " numbers\n";

                for(int i = 0; i < N; i++){
                        test2.push_back(static_cast <float> (rand()) / (static_cast <float> (RAND_MAX/LIM)));
                }

                start = clock();
                res = k_mins(test2, K, 256);
                end = clock();
                res.sort();

                cout << "\nExercise version time elapsed: "
                << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";
                for(auto i = res.begin(); i != res.end(); ++i) cout << *i << " ";

                start = clock();
                res = classic_k_mins(test2, K);
                end = clock();
                cout << "\nNormal version time elapsed: "
                << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";
                res.sort();
                for(auto i = res.begin(); i != res.end(); ++i) cout << *i << " ";
                cout << endl;
        }
        return 0;
}
```

Przeprowadzono testy polegające na znajdywaniu 10 najmniejszych liczb spośród 100, 1000, ..., 10000000 losowych. Ich wyniki poniżej.

Dla przedziału $[0, 0.3]$

```
Testing 10 min of 100 numbers

Exercise version time elapsed: 4.9e-05 ===========
0.000390449 0.000428517 0.000828413 0.00308767 0.00593011 0.00805869 0.0144824 0.0155922 0.0163935 0.0179536
Normal version time elapsed: 1.2e-05 ===========
0.000390449 0.000428517 0.000828413 0.00308767 0.00593011 0.00805869 0.0144824 0.0155922 0.0163935 0.0179536


Testing 10 min of 1000 numbers

Exercise version time elapsed: 0.000299 ===========
1.45443e-05 0.000410852 0.000478698 0.000584832 0.00064337 0.000650633 0.00110743 0.00139208 0.00250844
    0.00252012
```

```
Normal version time elapsed: 0.000139 ===========
1.45443e-05 0.000410852 0.000478698 0.000584832 0.00064337 0.000650633 0.00110743 0.00139208 0.00250844
    0.00252012


Testing 10 min of 10000 numbers

Exercise version time elapsed: 0.001904 ===========
3.08049e-06 7.87438e-06 3.88549e-05 4.16219e-05 0.000112928 0.000124436 0.000154647 0.000217448 0.000240922
    0.000289948
Normal version time elapsed: 0.001321 ===========
3.08049e-06 7.87438e-06 3.88549e-05 4.16219e-05 0.000112928 0.000124436 0.000154647 0.000217448 0.000240922
    0.000289948


Testing 10 min of 100000 numbers

Exercise version time elapsed: 0.02168 ===========
9.05385e-06 1.23579e-05 1.66085e-05 1.82872e-05 2.14633e-05 2.16095e-05 2.30055e-05 2.30833e-05 2.39168e-05
    3.25426e-05
Normal version time elapsed: 0.025814 ===========
9.05385e-06 1.23579e-05 1.66085e-05 1.82872e-05 2.14633e-05 2.16095e-05 2.30055e-05 2.30833e-05 2.39168e-05
    3.25426e-05


Testing 10 min of 1000000 numbers

Exercise version time elapsed: 0.19984 ===========
5.56698e-07 5.93299e-07 6.1132e-07 6.87176e-07 1.1398e-06 1.32071e-06 1.8985e-06 1.99224e-06 2.40044e-06
    2.80165e-06
Normal version time elapsed: 0.487195 ===========
5.56698e-07 5.93299e-07 6.1132e-07 6.87176e-07 1.1398e-06 1.32071e-06 1.8985e-06 1.99224e-06 2.40044e-06
    2.80165e-06


Testing 10 min of 10000000 numbers

Exercise version time elapsed: 1.97978 ===========
3.56231e-08 3.82774e-08 8.95467e-08 9.37376e-08 1.04913e-07 1.14273e-07 1.15531e-07 1.18464e-07 1.34809e-07
    1.66101e-07
Normal version time elapsed: 7.83855 ===========
3.56231e-08 3.82774e-08 8.95467e-08 9.37376e-08 1.04913e-07 1.14273e-07 1.15531e-07 1.18464e-07 1.34809e-07
    1.66101e-07
```

Dla przedziału $[0, 3]$

```
Testing 10 min of 100 numbers

Exercise version time elapsed: 0.000103 ===========
0.0275707 0.0491232 0.135877 0.178713 0.182404 0.183997 0.29177 0.301707 0.30237 0.305246
Normal version time elapsed: 2.2e-05 ===========
0.0275707 0.0491232 0.135877 0.178713 0.182404 0.183997 0.29177 0.301707 0.30237 0.305246
```

```
Testing 10 min of 1000 numbers

Exercise version time elapsed: 0.000472 ===========
9.49921e-05 0.00452236 0.0134568 0.0151314 0.0201986 0.0216173 0.0222465 0.0235811 0.0283786 0.0298431
Normal version time elapsed: 0.000218 ===========
9.49921e-05 0.00452236 0.0134568 0.0151314 0.0201986 0.0216173 0.0222465 0.0235811 0.0283786 0.0298431


Testing 10 min of 10000 numbers

Exercise version time elapsed: 0.004294 ===========
6.03818e-05 0.00107425 0.00146667 0.00167799 0.00218137 0.00232647 0.00249592 0.00284687 0.00301662 0.00310513
Normal version time elapsed: 0.002086 ===========
6.03818e-05 0.00107425 0.00146667 0.00167799 0.00218137 0.00232647 0.00249592 0.00284687 0.00301662 0.00310513


Testing 10 min of 100000 numbers

Exercise version time elapsed: 0.03344 ===========
1.62148e-05 1.96514e-05 6.79744e-05 7.62474e-05 0.000103887 0.000115382 0.000162771 0.000194962 0.000248772
    0.000289361
Normal version time elapsed: 0.033659 ===========
1.62148e-05 1.96514e-05 6.79744e-05 7.62474e-05 0.000103887 0.000115382 0.000162771 0.000194962 0.000248772
    0.000289361


Testing 10 min of 1000000 numbers

Exercise version time elapsed: 0.213393 ===========
5.45103e-06 7.52974e-06 7.53533e-06 8.5691e-06 1.22502e-05 1.5765e-05 2.1012e-05 2.40728e-05 3.69572e-05
    4.00711e-05
Normal version time elapsed: 0.496395 ===========
5.45103e-06 7.52974e-06 7.53533e-06 8.5691e-06 1.22502e-05 1.5765e-05 2.1012e-05 2.40728e-05 3.69572e-05
    4.00711e-05


Testing 10 min of 10000000 numbers

Exercise version time elapsed: 2.08863 ===========
8.03266e-07 1.95997e-06 2.09548e-06 2.16113e-06 2.4992e-06 2.80235e-06 3.41004e-06 4.02052e-06 4.22867e-06
    4.28315e-06
Normal version time elapsed: 8.704 ===========
8.03266e-07 1.95997e-06 2.09548e-06 2.16113e-06 2.4992e-06 2.80235e-06 3.41004e-06 4.02052e-06 4.22867e-06
    4.28315e-06
```