Sprawozdanie 1

Piotr Kotara

Marzec 2019

Rozdział 1

LAB 1

1.1 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++:

```cpp
#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.

1.2 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:

```python
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.

1.3 Sumowanie liczb pojedynczej precyzji w języku C:

1.3.1 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:

```cpp
#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;
}
```

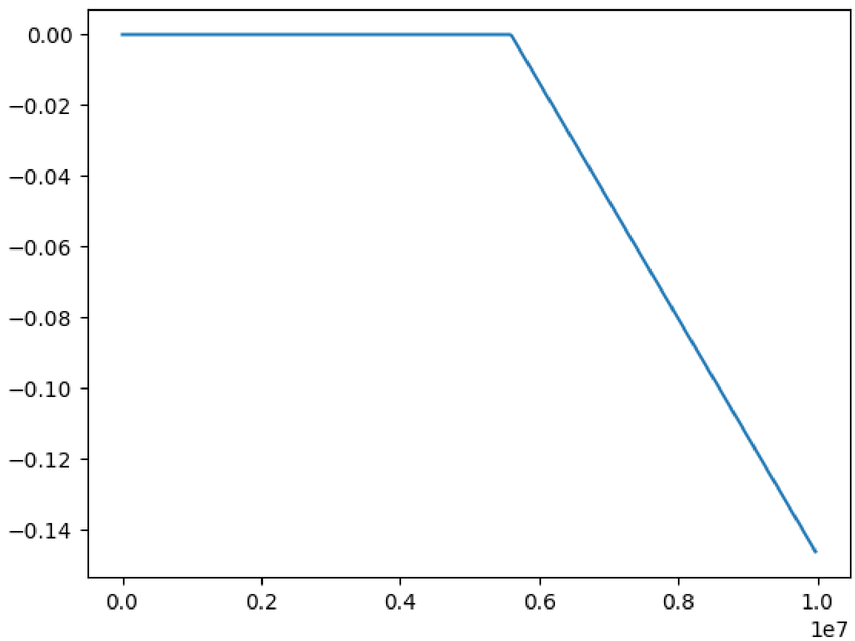
Wynik jest dokładny dla $v = 0.25$.

```
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 1.1: 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
```

1.4 Algorytm Kahana w języku C:

Kod:

```c
    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:

```cpp
#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);
}
```

1.5 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
```

Rozdział 2

LAB 1

2.1 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 rzęd 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.

2.2 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:

```python
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.

2.3 Sumowanie liczb pojedynczej precyzji w języku C:

2.3.1 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:

```cpp
#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;
}
```

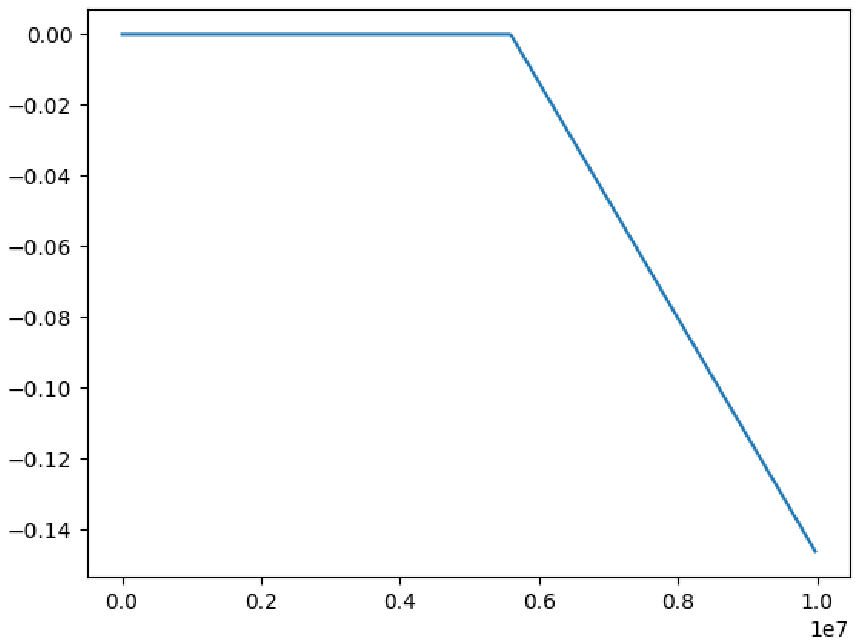
Wynik jest dokładny dla $v = 0.25$.

```
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 2.1: 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
```

2.4 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);
}
```

2.5 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
```

Rozdział 3

LAB 2

3.1 Napisz (Python)

```
import numpy as np

import time

def solve(matrix :np.ndarray, vector: np.ndarray):
    if(matrix.shape[1] != matrix.shape[1]) :
       print("macierz musi byc kwadratowa", matrix[1][0])
       exit()

    if(vector.shape[1] != 1 and vector.shape[0] != matrix.shape[0]) :
       print("macierz musi byc kwadratowa", matrix[1][0])
       exit()

    # print(matrix)
    # print(np.linalg.inv(matrix))
    # print("====================")
    size = matrix.shape[0]
    result = np.identity(size)

    for column in range(0, size):
        for row in range(column, size):
            if matrix.item((row, column)) != 0:
                matrix[[column, row]] = matrix[[row, column]]
                result[[column, row]] = result[[row, column]]                 # zamien zawsze tak, aby bu
        value = matrix.item((column, column))
        for row in range(0, size):
            if(row == column): continue
            item = matrix.item((row, column))
            times = (item/value)
            for cell in range(0, size):
                curr = matrix.item((row, cell))
                matrix.itemset((row, cell), curr - (matrix.item(column, cell)*times))

                curr = result.item((row, cell))
                result.itemset((row, cell), curr - (result.item(column, cell)*times))
```

```
    for column in range (0, size):
        value = matrix.item((column, column))
        for row in range(0, size):
            matrix.itemset((column, row), matrix.item((column, row)) / value)
            result.itemset((column, row), result.item((column, row)) / value)


    # print( matrix)
    # print(result)


    return np.matmul(result, vector)
        # for i in range(0, size):
        # for row in range(0, size):

#print(np.matrix('1 3 2 ; 1 3 3 ').shape)
#solve(np.matrix('0 2 0 4 ; 0 0 0 3 ; 3 2 0 2 ; 0 2 3 4'))

dim = 600
m = 1000 * np.random.rand(dim, dim)
v = 1000 * np.random.rand(dim, 1)
print(m)

start = time.time()
reslib = np.linalg.solve(m, v)
end = time.time()

libtime = end-start

start = time.time()
myres = solve(m, v)
end = time.time()

mytime = end-start


print(reslib - myres)

print("lib time:", libtime, "mytime: ", mytime)
```

Wyniki dla macierzy 600x600:

```
...
[ 1.32283628e-11]
[ 4.39959180e-12]
[ 7.21332022e-12]
[-2.13793983e-11]
[-2.40752140e-11]
[ 4.48685533e-12]
[-4.05706024e-12]
```

```
[-2.08548734e-11]]
lib time: 0.3574869632720947 mytime: 195.09919571876526
```

3.2 Napisz i sprawdź funkcję dokonującą faktoryzacji A=LU macierzy A. Zastosuj częściowe poszukiwanie elementu wiodącego oraz skalowanie.

```python
import numpy as np

def doolitleLU(matrix: np.ndarray):
    if(matrix.shape[0] != matrix.shape[1]) :
        print("macierz musi byc kwadratowa", matrix[1][0])
        exit()

    # print(matrix)
    # print(np.linalg.inv(matrix))
    # print("====================")
    size = matrix.shape[0]
    result = np.copy(matrix)

    L = np.ndarray(matrix.shape)
    U = np.ndarray(matrix.shape)

    for col in range(0, size):
        if(matrix.item((col, col)) == 0):
            for row in range(col, size):
                if(matrix.item((row, col)) != 0):
                    matrix[[col, row]] = matrix[[row, col]]
                    break

    for column in range(0, size):
        for row in range(column, size):
            sum = 0
            for i in range(0, column):
                sum += L.item((column, i)) * U.item((i, row))
            U.itemset((column, row), matrix.item(column, row) - sum)
        for row in range(column, size):
            if(column == row):
                L.itemset((column, column), 1)
            else:
                sum = 0
                for i in range(0, row):
                    sum+= L.item((row, i)) * U.item((i, column))
                L.itemset((row, column), ((matrix.item(row, column) - sum )/ U.item((column, column))))

    np.set_printoptions(suppress=True)

    print(matrix)
    np.set_printoptions(suppress=True)
    print(L)
    print(U)
```

```
np.set_printoptions(suppress=True)

doolitleLU(np.matrix('0 1 2 ; 4 5 6 ; 8 9 10'))
```

```
[[ 4  5  6]
[ 0  1  2]
[ 8  9 10]]
[[ 1.00000000e+000 6.94327138e-310 4.83245960e+276]
[-8.03704345e-095 1.00000000e+000 8.48585418e-096]
[ 2.00000000e+000 -1.00000000e+000 1.00000000e+000]]
[[4. 5. 6.]
[0. 1. 2.]
[0. 0. 0.]]
```

wynik dla macierzy L wynika z niedoskonałości arytmetyki komputerowej

3.3 Implementacja algorytmu Floyd-Warshall przy pomocy mnożenia macierzy. Język Python bądź C/C++ i prosty przykład z porównaniem wydajnościowym do wersji z pętlami mnożącymi. http://phdopen.mimuw.edu

```
import numpy as np
import math
import time

def min_plus(matrix: np.ndarray, other: np.ndarray):
    if(matrix.shape[0] != matrix.shape[1] != other.shape[0] != other.shape[1]) :
       print("macierz musi byc kwadratowa", matrix[1][0])
       exit()

    dim = matrix.shape[0]
    result = np.ndarray((dim, dim))

    for i in range (0, dim):
        for j in range (0, dim):
            minVal = matrix.item((i, 0)) + other.item((0, j))
            for k in range (1, dim):
                minVal = min(minVal, matrix.item(i, k) + other.item(k, j))
            result.itemset((i, j), minVal)

    return result

def shortest_path(matrix: np.ndarray):
    if(matrix.shape[0] != matrix.shape[1]) :
        print("macierz musi byc kwadratowa", matrix[1][0])
        exit()
    res = matrix
```

```
    for i in range(0, math.ceil(math.log2(matrix.shape[0]))):
        res = min_plus(res, res)
    return res

def floyd_warshall(matrix: np.ndarray):
    dim = matrix.shape[0]
    for k in range(0, dim):
        for i in range(0, dim):
            for j in range(0, dim):
                matrix.itemset((i, j), min(matrix.item((i, j)) , matrix.item((i, k))+ matrix.item((k, j))))

    return matrix

I = np.infty
m = np.array([[0, I, 1, I],
                [2, 0, 5, 1],
                [I, 2, 0, 1],
                [8, I, 12, 0]])
print(m)
mm = min_plus(m, m)
#print (mm)
print(shortest_path(m))
print(floyd_warshall(m))

test = np.random.rand(500, 500)

for i in range(0, 500):
    test.itemset((i, i), 0)

start = time.time()
floyd_warshall(test)
end = time.time()



libtime = end-start
start = time.time()
shortest_path(test)
end = time.time()


shorttime = end-start


print("lib time:", libtime, "mytime: ", shorttime)
```

```
lib time: 0.6981124877929688 mytime: 2.9570999145507812
```

3.4 Zaimplementuj 3 metody mnożenia macierzy rzadkich wedle formatów zaprezentowanych pod wskazanym linkiem: https://docs.nvidia.com/cuda/cu (COO/CSR/CSC/ELL) i porównaj jak ilość zer wpływa na czas mnożenia macierzy i ilość zużytej pamięci. Porównanie dokonaj do standardowej metody mnożenia macierzy o złożoności N^3. Język C/C++. Uszczegółowienie: można wykorzystać https://www.it.uu.se/education/phd_ 1 i można pokazać to na przykładzie mnożenia macierz-wektor.

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <ctime>
#include <cstdlib>


using namespace std;

class Matrix{
public:
    vector<vector<float>> data;
    size_t dimX, dimY;

    void setVal(size_t x, size_t y, float val){
        data[x][y] = val;
    }

    float getVal(size_t x, size_t y){
        return data[x][y];
    }

    vector<float> operator *(vector<float> vec){
        if(dimY != vec.size()) return vector<float>();

        vector<float> res;
        res.resize(dimY);
        for(int i = 0; i < dimY; i++){
            for(int j = 0; j < dimX; j++){
                res[i] += data[i][j] * vec[j];
            }
        }
        return res;
    }

};


struct Coord{
    size_t x, y;
    float val;
```

```
};

Coord getCoord(size_t y, size_t x, float val){
    Coord tmp;
    tmp.x = x;
    tmp.y = y;
    tmp.val = val;
    return tmp;
}

bool operator < (const Coord &c1, const Coord &c2) {
    if(c1.y < c2.y) return true;
    else return (c1.y == c2.y && c1.x < c2.x);
}

bool operator > (const Coord &c1, const Coord &c2) {
    if(c1.y > c2.y) return true;
    else return (c1.y == c2.y && c1.x > c2.x);
}


class COOMatrix{
public:

    size_t dimX, dimY;
    vector<Coord> data;

    COOMatrix (size_t x, size_t y){
        dimX = x;
        dimY = y;
    }

    COOMatrix (){
        dimX = 0;
        dimY = 0;
    }

    bool setVal(size_t x, size_t y, float val){
        if(x > dimX || y > dimY) return false;

        for (auto i = data.begin(); i != data.end(); ++i){
            if( i->x == x && i->y == y){
                if(val == 0){
                    data.erase(i);
                }
                else{
                    i->val = val;
                }
                return true;
            }
        }
        Coord tmp;
```

```
        tmp.x = x;
        tmp.y = y;
        tmp.val = val;
        data.push_back(tmp);
        sort(data.begin(), data.end());
    }


    vector<float> operator * (vector<float> &vec){

        if(this->dimY != vec.size()) return vector<float>();

        vector<float> result;
        result.resize(vec.size());


        for(Coord c : data){
            result[c.y] += c.val * vec[c.x];
        }

        return result;
    }

    // void print (){
    //     for(int i = 0; i < dimY; i++){
    //         for(int j = 0; j< dimX; j++){
    //             if(data)
    //         }
    //     }
    // }


};

struct CSRCoord{
    float val;
    size_t col;
};

CSRCoord getCrec(size_t col, float val){
    CSRCoord tmp;
    tmp.col = col;
    tmp.val = val;

    return tmp;
}

class CSRMatrix{
    public:

    size_t dimX, dimY;
    vector<CSRCoord> colTable;
```

```
    vector<size_t> offset;



    vector<float> operator * (vector<float> &vec){

        if(this->dimY != vec.size()) return vector<float>();

        vector<float> result;
        result.resize(vec.size());


        for(size_t i = 0; i < vec.size(); i++){
            for(size_t j = offset[i]; j < offset[i+1]; j++){
                result[i] += colTable[j].val * vec[colTable[j].col];
            }

        }



        return result;
    }

};

class ELLMatrix{
public:
    int MAX_ELEM_ROW;
    size_t dimX, dimY;

    vector<vector<float>> values;
    vector<vector<int>> columns;

    vector<float> operator * (vector<float> &vec){

        if(this->dimY != vec.size()) return vector<float>();

        vector<float> result;
        result.resize(vec.size());


        for(size_t i = 0; i < dimY; i++){
            for(size_t j = 0; j < MAX_ELEM_ROW; j++){
                if(columns[i][j] == -1) continue;
                result[i] += vec[columns[i][j]] * values[i][j];
            }
        }

        return result;
    }
```

```
};

void print_vec(vector<float> v){
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << " ";
    }
    cout << endl;
}

int main(){

    vector <float> v {1,2,5};
    Matrix m;
    m.dimX = 3;
    m.dimY = 3;
    m.data.push_back(vector<float>{3,0,2});
    m.data.push_back(vector<float>{0,1,2});
    m.data.push_back(vector<float>{2,0,1});


    print_vec(m * v);

    COOMatrix mo;
    mo.dimX = 3;
    mo.dimY = 3;
    mo.data.push_back(getCoord(0, 0, 3));
    mo.data.push_back(getCoord(0, 2, 2));
    mo.data.push_back(getCoord(1, 1, 1));
    mo.data.push_back(getCoord(1, 2, 2));
    mo.data.push_back(getCoord(2, 0, 2));
    mo.data.push_back(getCoord(2, 2, 1));

    print_vec(mo * v);

    CSRMatrix mc;
    mc.dimX = 3;
    mc.dimY = 3;
    mc.colTable.push_back(getCrec(0,3));
    mc.colTable.push_back(getCrec(2,2));
    mc.colTable.push_back(getCrec(1,1));
    mc.colTable.push_back(getCrec(2,2));
    mc.colTable.push_back(getCrec(0,2));
    mc.colTable.push_back(getCrec(2,1));

    mc.offset.push_back(0);
    mc.offset.push_back(2);
    mc.offset.push_back(4);
    mc.offset.push_back(6);
    print_vec(mc * v);

    ELLMatrix ml;
    ml.dimY = 3;
```

```
ml.dimX = 3;

ml.MAX_ELEM_ROW= 2;
ml.columns.push_back(vector<int>{0,2});
ml.columns.push_back(vector<int>{1,2});
ml.columns.push_back(vector<int>{0,2});

ml.values.push_back(vector<float>{3,2});
ml.values.push_back(vector<float>{1,2});
ml.values.push_back(vector<float>{2,1});

print_vec(ml * v);

srand(time(NULL));

const size_t DIMX = 6000;
const size_t DIMY = 6000;
const int  ZEROPROB = 80;

const float LO = 0;
const float HI = 1000;

Matrix testm;
testm.dimX = DIMX;
testm.dimY = DIMY;

testm.data.resize(DIMY);

float tmpV;

for(int i = 0; i < DIMX; i++){
    for(int j = 0; j < DIMY; j++){
        if(rand() % 100 < ZEROPROB){
            tmpV = 0;
        }
        else
        {
            tmpV = LO + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(HI-LO)));
        }
        testm.data[i].push_back(tmpV);
    }
}

vector<float> vec;
vec.resize(DIMY);
for(int i = 0; i <  DIMY; i ++){
    vec[i] = LO + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(HI-LO)));
}

clock_t start, end;

start = clock();
```

```
auto a = testm * vec;
end = clock();
a = a;

cout << "\nNormal version time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";

COOMatrix testo;
testo.dimX = DIMX;
testo.dimY = DIMY;
 for(int i = 0; i < DIMX; i++){
    for(int j = 0; j < DIMY; j++){
        if(rand() % 100 < ZEROPROB) continue;
        testo.data.push_back(getCoord(i, j, LO + static_cast <float> (rand()) /( static_cast <float> (RAN
    }
 }
start = clock();
auto ao = testo * vec;
end = clock();
ao= ao;

cout << "\nNormal version time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";

CSRMatrix testc;
testc.dimX = DIMX;
testc.dimY = DIMY;
int cntr = 0;
testc.offset.push_back(cntr);
 for(int i = 0; i < DIMY; i++){
    for(int j = 0; j < DIMX; j++){
        if(rand() % 100 < ZEROPROB) continue;
        testc.colTable.push_back(getCrec(j, LO + static_cast <float> (rand()) /( static_cast <float> (RAN
        cntr++;
    }
    testc.offset.push_back(cntr);
 }

 start = clock();
auto ac = testc * vec;
end = clock();
ac= ac;

cout << "\nNormal version time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";

ELLMatrix teste;
teste.dimX = DIMX;
teste.dimY = DIMY;
vector<int> tmpcol;
vector<float> tmpVal;
teste.MAX_ELEM_ROW = static_cast<int>(DIMX*ZEROPROB/100);
 for(int i = 0; i < DIMY; i++){
    for(int j = 0; j < teste.MAX_ELEM_ROW; j++){
        tmpVal.push_back(LO + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(HI-LO))));
```

```
            tmpcol.push_back(rand() % DIMX);
        }
        teste.columns.push_back(tmpcol);
        teste.values.push_back(tmpVal);
        tmpcol.clear();
        tmpVal.clear();
     }

    start = clock();
    auto ae = teste * vec;
    end = clock();
    ae= ae;

    cout << "\nELL    version time elapsed: " << (end-start)*1.0/CLOCKS_PER_SEC << " ===========\n";
```

```
Normal version time elapsed: 0.408693 ===========

COO version time elapsed: 0.112109 ===========

CSR version time elapsed: 0.094666 ===========

ELL   version time elapsed: 0.588516 ===========
```

Rozdział 4

LAB 3

4.1 Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody bisekcji funkcję f. Funkcja przyjmuje następujące argumenty: krańce przedziału, błąd bezwzględny obliczeń. Funkcja ma zwracać wyznaczone miejsce zerowe i liczbę iteracji potrzebną do uzyskania określonej dokładności. Przetestuj działanie metody dla funkcji podanych na początku instrukcji. Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody bisekcji funkcję f. Funkcja przyjmuje następujące argumenty: krańce przedziału, błąd bezwzględny obliczeń.

```python
def bisect(F, min:float, max:float, accuracy:float, itNo:int):
    center = (max+min)/2
    if(abs(F(center)) < accuracy ):
        print("Used iterations:", itNo)
        return center
    else:
        if(F(center)*F(min) < 0):
            return bisect(F, min, center, accuracy, itNo+1)
        else:
            return bisect(F, center, max, accuracy, itNo+1)

Fa = lambda x: math.cos(x) * math.cosh(x) - 1
Fb = lambda x: 1/x - np.tan(x) #[0, 2pi] z badania przebiegu zmienności wiemy że jedno zero jesst w [0.5, 1]
Fc = lambda x: 2**(-x) + math.exp(x) + 2 * math.cos(x) - 6
```

4.2 **Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody Newtona funkcję f. Funkcja ma wykorzystywać dwa kryteria stopu: maksymalną liczbę iteracji, moduł różnicy kolejnych przybliżeń mniejszy od danej wartości eps. Oprócz przybliżonej wartości pierwiastka funkcja ma zwrócić liczbę iteracji potrzebną do uzyskania określonej dokładności eps. Porównaj zbieżność metody ze zbieżnością uzyskaną dla metody bisekcji. Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody bisekcji funkcję f. Funkcja przyjmuje następujące argumenty: krańce przedziału, błąd bezwzględny obliczeń.**

```
def newton_solve(F, min:float, max:float, accuracy:float, itLim: int):
    x_last = np.infty
    x = min
    i = 0
    while(abs(x - x_last) > accuracy and i < itLim):
        x_last = x
        x = x - (F(x) / derivative(F, x, 1e-10))
        i+=1
    return x
```

4.3 **Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody Newtona funkcję f. Funkcja ma wykorzystywać dwa kryteria stopu: maksymalną liczbę iteracji, moduł różnicy kolejnych przybliżeń mniejszy od danej wartości eps. Oprócz przybliżonej wartości pierwiastka funkcja ma zwrócić liczbę iteracji potrzebną do uzyskania określonej dokładności eps. Porównaj zbieżność metody ze zbieżnością uzyskaną dla metody bisekcji. Napisz w Python bądź C/C++ funkcję rozwiązującą przy pomocy metody bisekcji funkcję f. Funkcja przyjmuje następujące argumenty: krańce przedziału, błąd bezwzględny obliczeń.**

```
def euler_solve(F, min:float, max:float, accuracy:float, itLim: int):
    x1 = min
    x2 = max
    i = 0
    while(abs(x2-x1) > accuracy and i < itLim):
        x3 = (F(x2)*x1-F(x1)*x2)/(F(x2)-F(x1))
        x1 = x2
        x2 = x3
        i+=1
    return x3
```

Rysunek 4.1: Wykres funkcji a

Rysunek 4.2: Wykres funkcji b

4.4 Podsumowanie

4.4.1 Wykresy funkcji

```
   FA=======
Used iterations bisect: 43
Bisect: 4.730040744862693
Used iterations Newton: 4
Newton: 4.730040744862704
Used iterations Euler : 7
Euler : 4.730040744862704
FB==(1)=====
Used iterations bisect: 38
Bisect: 0.8603335890193193
Used iterations Newton: 5
Newton: 0.8603335890193797
Used iterations Euler : 5
Euler : 0.8603335890193797
FB==(2)=====
Used iterations bisect: 37
Bisect: 3.425618459481484
Used iterations Newton: 5
Newton: 3.4256184594817283
Used iterations Euler : 7
Euler : 3.4256184594817283
FC========
Used iterations bisect: 40
Bisect: 1.8293836019338414
Used iterations Newton: 9
Newton: 1.829383601933849
Used iterations Euler : 11
Euler : 1.8293836019338487
```

Rysunek 4.3: Wykres funkcji c