Ćwiczenie 1

Ewelina Badeja

1. Treść zadania

4. Rozważmy funkcję $f(x) = (e^x - 1) / x$. Można wykazać (jak?), że $\lim_{x \to 0} f(x) = 1$
   a) Sprawdzić tę własność obliczając wartość funkcji dla $x = 10^{-k}$ dla $k = 1, \ldots, 15$. Czy rezultaty są zgodne z oczekiwaniami?
   b) Powtórzyć obliczenia tym razem korzystając ze wzoru:
   $$f(x) = (e^x - 1) / \ln(e^x) .$$
   Porównać rezultaty z tymi uzyskanymi w punkcie a). Spróbować wyjaśnić ewentualne różnice.

Obliczenia wykonać dla zmiennych typu *float, double, long double*. Zwrócić uwagę na typ argumentów i wyników dla funkcji bibliotecznych wykorzystywanych w obliczeniach.

2. Obliczanie granicy f(x)

$\lim_{x \to 0}(e^x - 1) = 0$ oraz $\lim_{x \to 0} x = 0$, zatem korzystając z reguły de l'Hospitala liczymy pochodne licznika i mianownika i otrzymujemy $\lim_{x \to 0} \frac{(e^x - 1)'}{x'} = \lim_{x \to 0}(\frac{e^x}{1}) = 1.$

3. Kod źródłowy

Żeby mieć jak największą kontrolę nad typem zmiennych używanych w obliczeniach, zdecydowałam się na pisanie w języku C++. Na początek napisałam funkcje podnoszące 10 do odpowiedniej potęgi dla zmiennych typu float, double i long double, ponieważ funkcja pow() przyjmuje i zwraca tylko zmienne typu double.

```cpp
#include <iostream>
#include <cmath>

float potega_f(int k){
    float pom=10;
    for (int i=0;i<k;i++){
        pom/=10;
    }
    return pom;
}

double potega_d(int k){
    double pom=10;
    for (int i=0;i<k;i++){
        pom/=10;
    }
    return pom;
}

long double potega_ld(int k){
    long double pom=10;
    for (int i=0;i<k;i++){
        pom/=10;
    }
    return pom;
}
```

Fragment kodu 1

Następnie napisałam funkcje obliczające wartość funkcji f(x) dla danego k (równanie z podpunktu *a)* nazwałam równaniem 1, a równanie z podpunktu *b)* równaniem 2). Liczbę *e* uzyskiwałam poprzez wywołanie funkcji exp(0). Wyrażenie e^x obliczałam z użyciem funkcji pow. Zarówno funkcja exp jak i pow zwracają typ double, więc wynik musiałam później rzutować na pożądany typ (jedynym sposobem uniknięcia tego, jaki udało mi się wymyślić, było znalezienie odpowiednio dokładnego przybliżenia liczby *e* i rzutowania go na odpowiednie typy zmiennych, tylko trzeba by przewidywać co to znaczy “odpowiednio dokładny”). Z funkcją log z biblioteki math nie było problemów, bo zwraca ona taki sam typ, jaki przyjmuje.

```
28  float rownanie1_float(int k){
29      float pom = potega_f(k);
30      return float((pow( X: exp( X: 1), Y: pom)-1)/pom);
31  }
32
33  float rownanie2_float(int k){
34      float pom = potega_f(k);
35      return float((pow( X: exp( X: 1), Y: pom)-1)/log( X: exp( X: pom)));
36  }
37
38  double rownanie1_double(int k){
39      double pom = potega_d(k);
40      return (pow( X: exp( X: 1), Y: pom)-1)/pom;
41  }
42
43  double rownanie2_double(int k){
44      double pom = potega_d(k);
45      return (pow( X: exp( X: 1), Y: pom)-1)/log( X: exp( X: pom));
46  }
47
48  long double rownanie1_long_double(int k){
49      long double pom = potega_ld(k);
50      return (long double)((pow( X: exp( X: 1), Y: pom)-1)/pom);
51  }
52
53  long double rownanie2_long_double(int k){
54      long double pom = potega_ld(k);
55      return (long double)((pow( X: exp( X: 1), Y: pom)-1)/log( X: exp( X: pom)));
56  }
```

Fragment kodu 2

Na koniec w funkcji main wywoływałam w pętlach funkcje dla odpowiednich k.

```cpp
int main() {
    std::cout<<"float, rownanie 1\n";
    for (int k=1;k<=15;k++){
        std::cout<<"k="<<k<<":\t"<<rownanie1_float(k)<<"\n";
    }
    std::cout<<"\ndouble, rownanie 1\n";
    for (int k=1;k<=15;k++){
        std::cout<<" k="<<k<<":\t"<<rownanie1_double(k)<<"\n";
    }

    std::cout<<"\nlong double, rownanie 1\n";
    for (int k=1;k<=15;k++){
        std::cout<<"k="<<k<<":\t"<<rownanie1_long_double(k)<<"\n";
    }

    std::cout<<"\nfloat, rownanie 2\n";
    for (int k=1;k<=15;k++){
        std::cout<<"k="<<k<<":\t"<<rownanie2_float(k)<<"\n";
    }
    std::cout<<"\ndouble, rownanie 2\n";
    for (int k=1;k<=15;k++){
        std::cout<<" k="<<k<<":\t"<<rownanie2_double(k)<<"\n";
    }

    std::cout<<"\nlong double, rownanie 2\n";
    for (int k=1;k<=15;k++){
        std::cout<<"k="<<k<<":\t"<<rownanie2_long_double(k)<<"\n";
    }
    return 0;
```

Fragment kodu 3

4. Analiza wyników

(Nie wstawiałam wyników do tabeli, ponieważ traciły wtedy resztki czytelności.)

- **Równanie 1**

```
float, rownanie 1
k=1:    1.71828186511993408203125
k=2:    1.05170917510986328125
k=3:    1.00501668453216552734375
k=4:    1.00050202178955078125
k=5:    1.00004994869232177734375
k=6:    1.00000500679016113281250
k=7:    1.00000047683715820312500
k=8:    1
k=9:    1
k=10:   1
k=11:   1.00000023841857910156250
k=12:   1.00000011920928955078125
k=13:   1.00000894933929443359375
k=14:   0.99920082092285156250
k=15:   0.99920082092285156250
```

```
double, rownanie 1
 k=1:   1.71828182845904509079559829842764884233474731445312 5
 k=2:   1.05170918075647712441877956734970211982727050781 25
 k=3:   1.00501670841679491275044711073860526084899902343 75
 k=4:   1.00050016670838459731385228224098682403564453125
 k=5:   1.00005000166714097531439620070159435272216796875
 k=6:   1.00000500000696490587870357558131217956542968 75
 k=7:   1.00000049996218343117959648225223645567893981933 59375
 k=8:   1.00000004943368003829107237834250554442405700683 59375
 k=9:   0.99999999392252880703324535716092213988304138183 59375
 k=10:  1.00000008274037077704576859105145558714866638183 59375
 k=11:  1.00000008274037077704576859105145558714866638183 59375
 k=12:  1.00000008274037077704576859105145558714866638183 59375
 k=13:  1.00008890058234078956900248158490285277366638183 59375
 k=14:  0.99920072216264077535896603876608423888683319091796875
 k=15:  0.99920072216264077535896603876608423888683319091796875

long double, rownanie 1
k=1:    1.71828182845904509079559829842764884233474731445312 5
k=2:    1.05170918075647712441877956734970211982727050781 25
k=3:    1.00501670841679491275044711073860526084899902343 75
k=4:    1.00050016670838459731385228224098682403564453125
k=5:    1.00005000166714097542281641795014479612291324883699417114257812 5
k=6:    1.00000500000696490598712379282986262296617496758699417114257812 5
k=7:    1.00000049996218365333262162453209498380601871758699417114257812 5
k=8:    1.00000004943368026044409752062236407255113590508699417114257812 5
k=9:    0.99999999392252902913206039081650544630974764004349708557128906 25
k=10:   1.00000008274037099919879373333131411527574528008699417114257812 5
k=11:   1.00000008274037099919879373333131411527574528008699417114257812 5
k=12:   1.00000008274037099919879373333131411527574528008699417114257812 5
k=13:   1.00008890058234101172202762386476138090074528008699417114257812 5
k=14:   0.99920072216264088643547860990601350295037264004349708557128906 25
k=15:   0.99920072216264088648968871853028872465074528008699417114257812 5
```

Wszystkie typy zmiennych na początku zbliżały się do 1, jednak co zaskakujące, dla k=14 i dla k=15 dały wynik mniejszy od 1. W przypadku typu float najłatwiej można zaobserwować, że od pewnego momentu (k=10) funkcja rzekomo najpierw rośnie, a potem maleje (co nie może być prawdą, bo jest to funkcja monotoniczna). Dodatkowo można zauważyć, że long double dał dokładniejsze wyniki niż double (tj. dla większych k na pewno niepoprawne, ale z większą liczbą miejsc po przecinku).

- **Równanie 2**

```
float, rownanie 2
k=1:    1.7182818651199340820 3125
k=2:    1.051709175109863281 25
k=3:    1.0050166845321655273 4375
k=4:    1.000500202178955078 125
k=5:    1.0000499486923217773 4375
k=6:    1.0000050067901611328 125
k=7:    1.000000476837158203 125
k=8:    1
k=9:    1
k=10:   1
k=11:   1
k=12:   1
k=13:   1
k=14:   1
k=15:   1
```

```
double, rownanie 2
 k=1:   1.7182818284590450907955982984276488423347473144531 25
 k=2:   1.0517091807564762362403598672244697809219360351562 5
 k=3:   1.0050167084168057929360884372727014124393463134765625
 k=4:   1.0005001667083415206604968261672183871269226074218 75
 k=5:   1.0000500016667082103793973146821372210979461669921875
 k=6:   1.0000050000166667008016929685254581272602081298828125
 k=7:   1.0000005000001666033426772628445178270339965820312 5
 k=8:   1.0000000500000016945278957791742868721485137939453125
 k=9:   1.0000000049999999696126451453892514109611511230468 75
 k=10:  1.0000000005000000413701854995451867580413818359375
 k=11:  1.0000000000500000041370185499545186758041381835937 5
 k=12:  1.0000000000050000004137018549954518675804138183593 75
 k=13:  1.0000000000005000444502911705058068037033081054687 5
 k=14:  1.0000000000000499600361081320443190634250640869140625
 k=15:  1.0000000000000051070259132757200859487056732177734375
```

```
long double, rownanie 2
k=1:    1.7182818284590450907955982984276488423347473144531 25
k=2:    1.0517091807564762362403598672244697809219360351562 5
k=3:    1.0050167084168057929360884372727014124393463134765625
k=4:    1.0005001667083415206604968261672183871269226074218 75
k=5:    1.0000500016667082103793973146821372210979461669921875
k=6:    1.0000050000166667008016929685254581272602081298828125
k=7:    1.0000005000001666033426772628445178270339965820312 5
k=8:    1.0000000500000016945278957791742868721485137939453125
k=9:    1.0000000049999999696126451453892514109611511230468 75
k=10:   1.0000000005000000413701854995451867580413818359375
k=11:   1.0000000000500000041370185499545186758041381835937 5
k=12:   1.0000000000050000004137018549954518675804138183593 75
k=13:   1.0000000000005000444502911705058068037033081054687 5
k=14:   1.0000000000000499600361081320443190634250640869140625
k=15:   1.0000000000000051070259132757200859487056732177734375
```

Wyniki dla równania 2 są dużo bardziej wiarygodne, ponieważ wszystkie wyniki są malejące, jednak double dał takie same przybliżenia jak long double.

5. Wnioski

Oba równania, które zbiegają do 1, dały różne wyniki. Nie potrafię sensownie wytłumaczyć, dlaczego tak się stało (podejrzanymi są biblioteki i kompilator). Przydatną informacją, jaką udało mi się zdobyć podczas tego ćwiczenia, jest to, że większość funkcji matematycznych obsługuje typ double i często nie ma znaczenia, czy użyjemy double czy long double- wynik może być podobny.

6. **Dane techniczne**
    - Język programowania: C++
    - Kompilator: g++ (toolchain: MinGW)
    - System: Windows Server 2019
    - Procesor: Intel Core i7-7700HQ 2.8 GHz