第一次上机作业

问题叙述：

分别以单精度和双精度数据类型计算:

- n=10000 时 $f(n)=\sum_{i=1}^{n}\frac{1}{i^4}$ 的值，分别按从 i=1 到 10000 和 i=10000 到 1 的顺序计算，对比误差并分析结果。
- 已知 x=100000，分别采用以下公式计算 y，哪个结果更加准确？为什么？

$$y=\ln(x-\sqrt{x^2-1}) \tag{1}$$

$$y=-\ln(x+\sqrt{x^2-1}) \tag{2}$$

matlab 程序：

第一问：

```
%%%problem 1
clc, clear, close all
real = pi^4/90;
%%with single-precision data and forward summation
sum = 0;
for i = 1:10000
sum = sum + single(1/(i^4));
end
sum, %print
error = abs((sum - real)/real) %calculate error
%%with single-precision data and backward summation
sum = 0;
for i = 10000:-1:1
sum = sum + single(1/(i^4));
end
sum, %print
error = abs((sum - real)/real) %calculate error
%%with double-precision data and forward summation
sum = 0;
for i = 1:10000
sum = sum + 1/(i^4);
end
sum, %print
error = abs((sum - real)/real) %calculate error
%%with double-precision data and backward summation
```

```
sum = 0;
for i = 10000:-1:1
sum = sum + 1/(i^4);
end
sum, %print
error = abs((sum - real)/real) %calculate error
```

真值及误差计算:

```
from decimal import *
from turtle import end_fill
getcontext().prec = 100
sum = Decimal('0')
for i in range(1, 101, 1):
    sum = sum+Decimal('1')/(Decimal(str(i))*Decimal(str(i))
                            * Decimal(str(i))*Decimal(str(i)))
result = [Decimal('1.0823221'), Decimal('1.0823232'), Decimal(
    '1.082323233710861'), Decimal('1.082323233710805')]
error = [0, 0, 0, 0]
for i in range(4):
    error[i] = abs((result[i]-sum)/sum)
print(sum)
print(error)
```

程序运行结果:

	计算结果	对理论值误差	相对误差
单精度正向	1.0823221	1.0283846e-06	7.44088912e-07
单精度反向	1.0823232	3.7106314e-08	2.72243639e-07
双精度正向	1.082323233710861	2.558289554517808e-13	3.03390408e-07
双精度反向	1.082323233710805	3.077333064776834e-13	3.03390356e-07

第二问:

```
%%%problem 2
clc, clear, close all
x = 100000;
```

```
%%with double-precision data and algorithm 1
y = log(x - sqrt(x^2 - 1))
%%with double-precision data and algorithm 2
y = -log(x + sqrt(x^2 - 1))
%%with single-precision data and algorithm 1
y = log(x - single(sqrt(x^2 - 1)))
%%with double-precision data and algorithm 2
y = -log(x + single(sqrt(x^2 - 1)))
```

真值及误差计算:

```
from decimal import *
from turtle import end_fill
x = Decimal('100000')
getcontext().prec = 100
sum = -Decimal.ln(x+Decimal.sqrt(x*x-Decimal('1')))
print(sum)
result = [Decimal('-12.2060728'), Decimal('
    -12.206073762186564'), Decimal(
     '-12.206072645505174')]
error = [0, 0, 0]
for i in range(3):
    error[i] = abs((result[i]-sum)/sum)
print(sum)
print(error)
```

程序运行结果:

	计算结果	相对误差
单精度算法 1	-Inf	Inf
单精度算法 2	-12.2060728	1.26572101e-08
双精度算法 1	-12.206073762186564	9.14857238e-08
双精度算法 2	-12.206072645505174	2.21605881e-17

心得体会:

对于第一问，首先在精度固定的前提下考虑 i 从 1 加到 10000 和从 10000 加到 1。在大数与小数相加时，会产生误差，导致小数宛如“没加”。例如，一个小数 0.0010 和一个大数 4000 相加，使用一个假想的计算机，具有 4 位尾数和 1 位指数，对较小的数进行调整，使其指数与较大的数相匹配。将结果进行舍去处理得 $0.4000 * 10^4$ 而在无穷级数求和时，如果 i 从 1 加到 10000，初始项通常大于后面

的项这会导致 i 较大时产生较多的大数加小数误差。如果反向求和，即以升序而不是降序对级数求和则加的项较多之后大数加小数的问题会减缓，这种误差就会相应减小。

对于第二问，两种不同的计算方法若不考虑精度在数学上是等价的。然而，计算时会产生减性抵销，即两个几乎相等的浮点数相减时所引起的舍入误差。式 (1) 中 x 与 $\sqrt{x^2-1}$ 在 x 较大时几乎相等，会产生较为严重的减性抵消误差。若使用式 (2) 进行计算，则减性抵消误差就会大大消除。同时由于加号两端式子几乎相等，因此计算误差 (2) 会比 (1) 小。

同时题目要求分别以单精度和双精度进行计算。在其他条件相同时，双精度会比单精度计算结果更为精确。

相对误差计算公式：

$$\varepsilon = \frac{\text{计算结果} - \text{理论值}}{\text{理论值}} * 100\% \tag{3}$$

第一问的理论值为 $\frac{\pi^4}{90}$。利用 python 中的 decimal 库，可以精确地计算出真值（精度为 100 位）为 1.0823229053444731912938314576599521286104957347737161321181856369943998733840006543582762063085613 56。利用相对误差公式来比较误差大小，从而判断算法的优劣程度。由程序运行结果可知，单精度误差比双精度大，同一精度下正向误差远比反向误差大。但在单精度中反向算法优势更明显，这是因为双精度由于大数加小数的舍入误差对结果影响不大，两者误差基本相等。

第二问利用 python 可计算出真值为-12.20607264550517372950625189487994552274767082620203655486023616219909521052689052440120670600785998。单精度算法 1 直接无法计算出结果，这是因为产生了完全的减性抵消，ln 括号内的数变成了 0，该算法无法得出结果。由程序运行结果可知，单精度算法比双精度误差大，而同一种精度下算法 1 误差远大于算法 2。