第一章作业

问题叙述：

分别以单精度和双精度数据类型采用以下公式分别计算 ln2 的近似值：

$$\ln(1+x)=\sum_{n=1}^{+\infty}(-1)^{n+1}\frac{x^n}{n},(-1<x\leq 1) \tag{1}$$

$$\ln\frac{1+x}{1-x}=2(x+\frac{x^3}{3}+\frac{x^5}{5}+\cdots+\frac{x^{2n+1}}{2n+1}+\cdots),(-1<x<1) \tag{2}$$

- 要求结果具有 4 位有效数字；
- 尝试得到更高精度的结果；
- 对结果进行分析、讨论。

问题分析：

对于第一问，定义：设 x^* 是 x 的一个近似数，表示为 $x^*=\pm 10^k\times 0.a_1a_2\cdots a_n$ 其中每个 a_i 均为 0，1，2，……，9 中的一个数字，且 $a_1\neq 0$，如果

$$|x-x^*|\leq\frac{1}{2}\times 10^{k-n} \tag{3}$$

则称 x^* 近似 x 有 n 位有效数字。

如果 x^* 具有 n 位有效数字，则其相对误差限为:

$$\delta_r(x^*)\leq\frac{1}{2a_1}\times 10^{-(n-1)} \tag{4}$$

由于题目要求有效数字为 4 位，则误差限 $\varepsilon_s=(0.5\times 10^{2-4})\%=0.5\times 10^{-4}$。

当前迭代结果的误差：

$$\varepsilon_a=\frac{\text{当前近似值}-\text{前一近似值}}{\text{当前近似值}}\times 100\% \tag{5}$$

采用绝对值（不考虑正负号），计算终止条件为判断当前迭代结果小于预先设定好的容限：

$$|\varepsilon_a|\leq\varepsilon_s \tag{6}$$

对于第二问，需要先计算出单精度数据类型机器精度，测试机器精度的方法为：不断迭代，寻找满足 $1+\varepsilon>1$ 的最小浮点数。

伪代码如下：

```
1 Epsilon = 1
2 DO
3     If(Epsilon+1<=1)EXIT
4     Epsilon = Epsilon/2
5 End DO
6 Epsilon = 2 * Epsilon
```

算法设计：

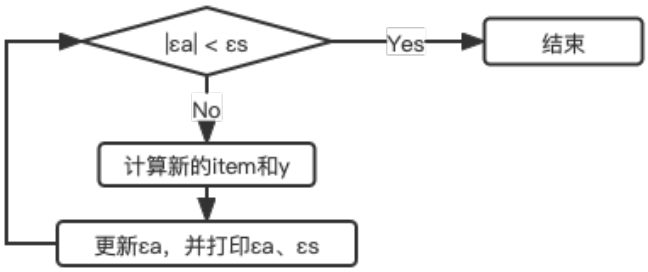

图 1: 算法设计

测试精度为机器精度时，将判断条件改为新加项小于机器精度即可。

第一问：

matlab 程序：

```
%%%problem 1
clc,clear,close all
real=log(2);
%%with single-precision data and algorithm 1
x=1;
sum=0;
i=1;
e_a=1;
while e_a>=0.5*0.0001
    sum=single(sum+(-1)^(i+1)*single(x)^i/i);
    if(i>=2)
        e_a=abs((sum_1(i-1)-sum)/sum);
        ea_1(i)=e_a;
    end
    es_1(i)=abs((sum-real)/real);
    sum_1(i)=sum;
    i=i+1;
end
sum,
%%with double-precision data and algorithm 1
x=1;
sum=0;
i=1;
```

```
e_a=1;
while e_a>=0.5*0.0001
    sum=sum+(-1)^(i+1)*x^i/i;
    if(i>=2)
        e_a=abs((sum_2(i-1)-sum)/sum);
        ea_2(i)=e_a;
    end
    es_2(i)=abs((sum-real)/real);
    sum_2(i)=sum;
    i=i+1;
end
sum,
%%with single-precision data and algorithm 2
x=1/3;
sum=0;
i=1;
e_a=1;
while e_a>=0.5*0.0001
    sum=single(sum+2*single(x)^i/i);
    if(i>=2)
        e_a=abs((sum_3((i+1)/2-1)-sum)/sum);
        ea_3((i+1)/2)=e_a;
    end
    es_3((i+1)/2)=abs((sum-real)/real);
    sum_3((i+1)/2)=sum;
    i=i+2;
end
sum,
%%with double-precision data and algorithm 2
x=1/3;
sum=0;
i=1;
e_a=1;
while e_a>=0.5*0.0001
    sum=sum+2*x^i/i;
    if(i>=2)
        e_a=abs((sum_4((i+1)/2-1)-sum)/sum);
```

```
61              ea_4((i+1)/2)=e_a;
62          end
63          es_4((i+1)/2)=abs((sum-real)/real);
64          sum_4((i+1)/2)=sum;
65          i=i+2;
66      end
67      sum,
```

程序运行结果:

表 1: 单精度算法 1

序号	当前计算值 y	近似百分比相对误差 ε_a	真百分比相对误差 ε_s
1	1	Inf	0.4426951
2	0.5000000	1	0.2786525
3	0.8333334	0.4000000	0.2022459
⋮	⋮	⋮	⋮
28848	0.6931221	5.0048759e-05	3.6113608e-05
28849	0.6931568	5.0046256e-05	1.3933342e-05
28850	0.6931221	5.0048759e-05	3.6113608e-05
28851	0.6931568	5.0046256e-05	1.3933342e-05
28852	0.6931222	4.9962760e-05	3.6027617e-05

表 2: 双精度算法 1

序号	当前计算值 y	近似百分比相对误差 ε_a	真百分比相对误差 ε_s
1	1	Inf	0.442695040888963
2	0.500000000000000	1	0.278652479555518
3	0.833333333333333	0.400000000000000	0.202245867407469
⋮	⋮	⋮	⋮
28851	0.693164510681439	5.000377704106826e-05	2.500208033702421e-05
28852	0.693129851039127	5.000454425757005e-05	2.500121374624657e-05
28853	0.693164509480190	5.000031102185020e-05	2.500034730074432e-05
28854	0.693129852240292	5.000107813183374e-05	2.499948083009518e-05
28855	0.693164508279107	4.999684548311008e-05	2.499861450488159e-05

表 3: 单精度算法 2

序号	当前计算值 y	近似百分比相对误差 ε_a	真百分比相对误差 ε_s
1	0.6666667	Inf	0.0382033
2	0.6913580	0.0357143	0.0025812
3	0.6930041	0.0023753	2.0637643e-04
4	0.6931348	1.8849636e-04	1.7883449e-05
5	0.6931460	1.6252387e-05	1.6310872e-06

表 4: 双精度算法 2

序号	当前计算值 y	近似百分比相对误差 ε_a	真百分比相对误差 ε_s
1	0.666666666666667	Inf	0.038203306074024
2	0.691358024691358	0.035714285714286	0.002581206298988
3	0.693004115226337	0.002375296912114	2.063996473191867e-04
4	0.693134757332288	1.884800965017501e-04	1.792292893281146e-05
5	0.693146047390827	1.628813809361991e-05	1.634817467324221e-06

更高精度的分析:

对于两种不同的算法，收敛速度越快，收敛次数越少，舍入误差积累的量就越少。由第一问可知算法二的收敛速度明显快于算法一且误差远小于算法一，因此在此问中采用算法二。首先计算出机器精度，随后将 ε_s 设定为机器精度，进行迭代计算。

matlab 程序:

```
%%%problem 2
clc,clear,close all
real=log(2);
%%with single-precision data and algorithm 2
%Calculate machine accuracy
epsilon=1;
while epsilon+1>1
    epsilon=single(epsilon/2);
end
epsilon=epsilon*2;
x=1/3;
sum=0;
i=1;
```

```
e_a=1;
while e_a>=epsilon
    sum=single(sum+2*single(x)^i/i);
    if(i>=2)
        e_a=abs((sum_1((i+1)/2-1)-sum)/sum);
        ea_1((i+1)/2)=e_a;
    end
    es_1((i+1)/2)=abs((sum-real)/real);
    sum_1((i+1)/2)=sum;
    i=i+2;
end
sum,
%%with double-precision data and algorithm 2
epsilon=1;
while epsilon+1>1
    epsilon=epsilon/2;
end
epsilon=epsilon*2;
x=1/3;
sum=0;
i=1;
e_a=1;
while e_a>=epsilon
    sum=sum+2*x^i/i;
    if(i>=2)
        e_a=abs((sum_2((i+1)/2-1)-sum)/sum);
        ea_2((i+1)/2)=e_a;
    end
    es_2((i+1)/2)=abs((sum-real)/real);
    sum_2((i+1)/2)=sum;
    i=i+2;
end
sum,
```

利用 python 进行真值计算和误差分析：

```
from decimal import *
from turtle import end_fill
```

```
x = Decimal('100000')
getcontext().prec = 100
sum = Decimal.ln(Decimal('2'))
print(sum)

result_single = Decimal('0.6931472')
error_single = abs((result_single-sum)/sum)

result_double = Decimal('0.693147180559945')
error_double = abs((result_double-sum)/sum)

print('single error')
print(error_single)
print('double error')
print(error_double)
```

程序运行结果：

表 5: 计算结果及误差

单精度	单精度误差	双精度	双精度误差
0.6931472	2.8046070e-08	0.693147180559945	4.463947063472170e-16

小结：

对于第一问，计算结果的真值可取 0.693147180559945，由计算结果可见，不论那种精度的哪种算法，当计算结束时得出的值前四位有效数字已与真值一样，说明了算法的有效性。同时，由于第一种算法的的项一正一负，收敛速度明显低于第二种算法，因此用第二种算法可以更高效地得出结果。

计算结果显示近似误差在刚达到 0.5e-04 以下时，实际误差却已经小于 0.5e-04。通过改变 ε_s 的大小理应可以调节结果的精度。

对于第二问，利用 python 进行真值计算，精确度为小数点后 100 位，我将这个结果当做真值进行误差分析。有计算结果可知，在使用机器精度为限定条件且在双精度计算条件下，16 次计算便收敛到了一个非常精确的值，其相对误差为 e-16 数量级，远低于第一问中的 e-06 数量级。因此，成功提升了计算精度。