第一章作业

一、问题叙述

分别以单精度和双精度数据类型用以下近似算法分别计算 p 的近似值：

$$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots\right) \qquad \pi = 6\left(0.5+\frac{0.5^3}{2\times3}+\frac{3\times0.5^5}{2\times4\times5}+\frac{3\times5\times0.5^7}{2\times4\times6\times7}+\cdots\right)$$

（1）假定真值未知，要求结果具有至少 4 位有效数字，给出计算结果。

（2）如果采用单精度数据类型要求计算结果达到机器精度，此时结果如何？采用双精度数据类型达到单精度机器精度要求以及更高的精度要求，计算结果如何？（测试机器精度：满足 1+e>1 的最小浮点数）

二、问题分析

本题应采用迭代的方法求解，当前迭代结果对真值的误差为：

$$\varepsilon_r = \frac{真值-近似值}{真值}\times 100\%$$

由于真值未知，所以用近似百分比误差估计值来衡量计算值与真值的接近程度：

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

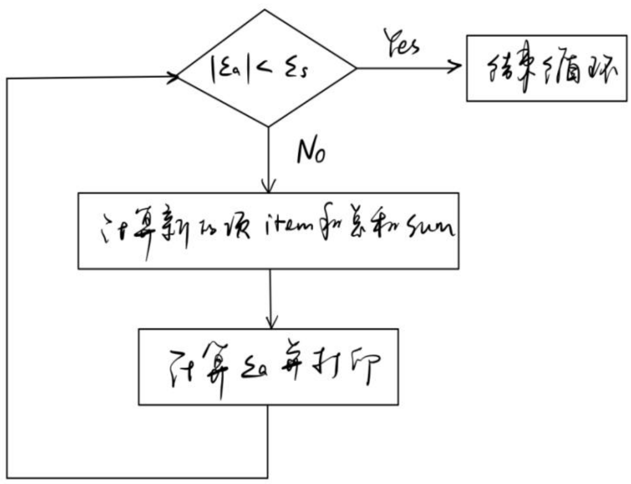
再有结果需保留四位有效数字，可得容限为：

$$\varepsilon_s = (0.5\times 10^{2-4})\% = 0.005\%$$

因此循环的终止条件为：$|\varepsilon_a| < \varepsilon_s$

三、算法设计

对于两种不同的计算方法，添加的新项都具有递推关系，因此设计算法如右图所示：



四、代码及运算结果

1、 $$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots\right)$$

（1）双精度运算，代码如下：

```
%using method one to calculate the approximation of pi, with double precision
close all;clear;clc;
n = 1 ;                                  %set 'n' as the number of terms
item = (-1)^(n+1) * 4 * (1/(2*n-1));     %calculate the first item
sum = item;                              %initialize 'sum'
e_a = inf;                               %initialize the error of current iteration results
e_s = 0.00005;                           %set the boundary
while abs(e_a) > e_s
    n = n + 1;                           %renew the counter
    item = (-1)^(n+1) * 4 * (1/(2*n-1)); %calculate the new item
    sum = sum + item;                    %renew the result
    e_a = item / sum;                    %renew the error
end

fprintf('n=%d  sum=%e  error_a=%e\t\n', n, sum, e_a);
```

运算结果如下：

```
n=12733   sum=3.141671e+00   error_a=4.999834e-05
```

（2）单精度运算，代码如下：

```
%using method one to calculate the approximation of pi, with single precision
close all;clear;clc;
n = 1;
item = single ((-1)^(n+1) * 4 * (1/(2*n-1)));
sum = single (item);
e_a = single (inf);
e_s = single (0.00005);
while abs(e_a) > e_s
    n = n + 1;
    item = single ((-1)^(n+1) * 4 * (1/(2*n-1)));
    sum = single(sum + item);
    e_a = single(item / sum);
end

fprintf('n=%d  sum=%e  error_a=%e\t\n', n, sum, e_a);
```

运算结果如下：

```
n=12733   sum=3.141679e+00   error_a=4.999822e-05
```

（3）改用单精度的机器精度作为循环终止条件时，代码如下：

```
%using method one to calculate the approximation of pi, with single precision
close all;clear;clc;
n = 1;
item = single ((-1)^(n+1) * 4 * (1/(2*n-1)));
sum = single (item);
e_a = single (inf);
e = eps(single(1));                          %change the boundary
while abs(e_a) > e
    n = n + 1;
    item = single ((-1)^(n+1) * 4 * (1/(2*n-1)));
    sum = single(sum + item);
    e_a = single(item / sum);
end

fprintf('n=%d  sum=%e  error_a=%e\t\n', n, sum, e_a);
```

运算结果如下：

```
n=5340347   sum=3.141597e+00   error_a=1.192093e-07
```

而改用双精度的机器精度作为循环终止条件时没有运行出结果。

2、 $$\pi = 6\left(0.5 + \frac{0.5^3}{2\times 3} + \frac{3\times 0.5^5}{2\times 4\times 5} + \frac{3\times 5\times 0.5^7}{2\times 4\times 6\times 7} + \cdots\right)$$

（1）双精度运算，代码如下：

```
%using method two to calculate the approximation of pi, with double precision
close all;clear;clc;
n = 1;
temp1 = double_factorial(2*n-3);
temp2 = double_factorial(2*n-2);
item = 6 * ( temp1*0.5^(2*n-1) / (temp2*(2*n-1)) );
sum = item;
e_a = inf;
e_s = 0.00005;
fprintf('n=%d\tsum=%e\terror_a=%e\t\t\t\titem=%e\n', n, sum, e_a, item);

while abs(e_a) > e_s
    n = n + 1;
    temp1 = double_factorial(2*n-3);
    temp2 = double_factorial(2*n-2);
    item = 6 * ( temp1*0.5^(2*n-1) / (temp2*(2*n-1)) );
    sum = sum + item;
    e_a = item / sum;
    fprintf('n=%d\tsum=%e\terror_a=%e\titem=%e\n', n, sum, e_a, item);
end
```

运算结果如下：

```
n=1  sum=3.000000e+00   error_a=Inf            item=3.000000e+00
n=2  sum=3.125000e+00   error_a=4.000000e-02   item=1.250000e-01
n=3  sum=3.139063e+00   error_a=4.479841e-03   item=1.406250e-02
n=4  sum=3.141155e+00   error_a=6.661988e-04   item=2.092634e-03
n=5  sum=3.141511e+00   error_a=1.133335e-04   item=3.560384e-04
n=6  sum=3.141577e+00   error_a=2.086323e-05   item=6.554343e-05
```

（2）单精度运算结果与双精度几乎完全一样：

```
n=1  sum=3.000000e+00   error_a=Inf            item=3.000000e+00
n=2  sum=3.125000e+00   error_a=4.000000e-02   item=1.250000e-01
n=3  sum=3.139062e+00   error_a=4.479841e-03   item=1.406250e-02
n=4  sum=3.141155e+00   error_a=6.661988e-04   item=2.092634e-03
n=5  sum=3.141511e+00   error_a=1.133335e-04   item=3.560384e-04
n=6  sum=3.141577e+00   error_a=2.086323e-05   item=6.554343e-05
```

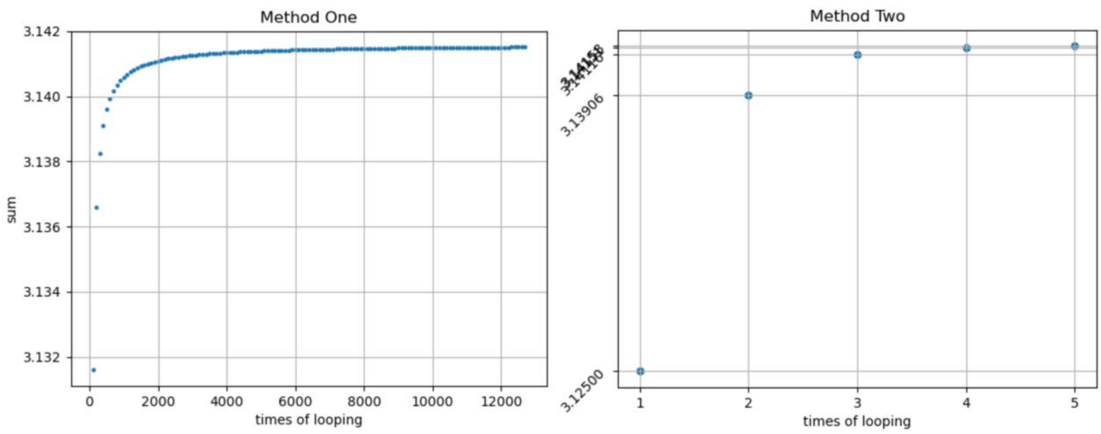
（3）分别以单精度和双精度的机器精度作为循环终止条件，结果如下：

```
n=10    sum=3.141592e+00    error_a=3.555928e-08
```

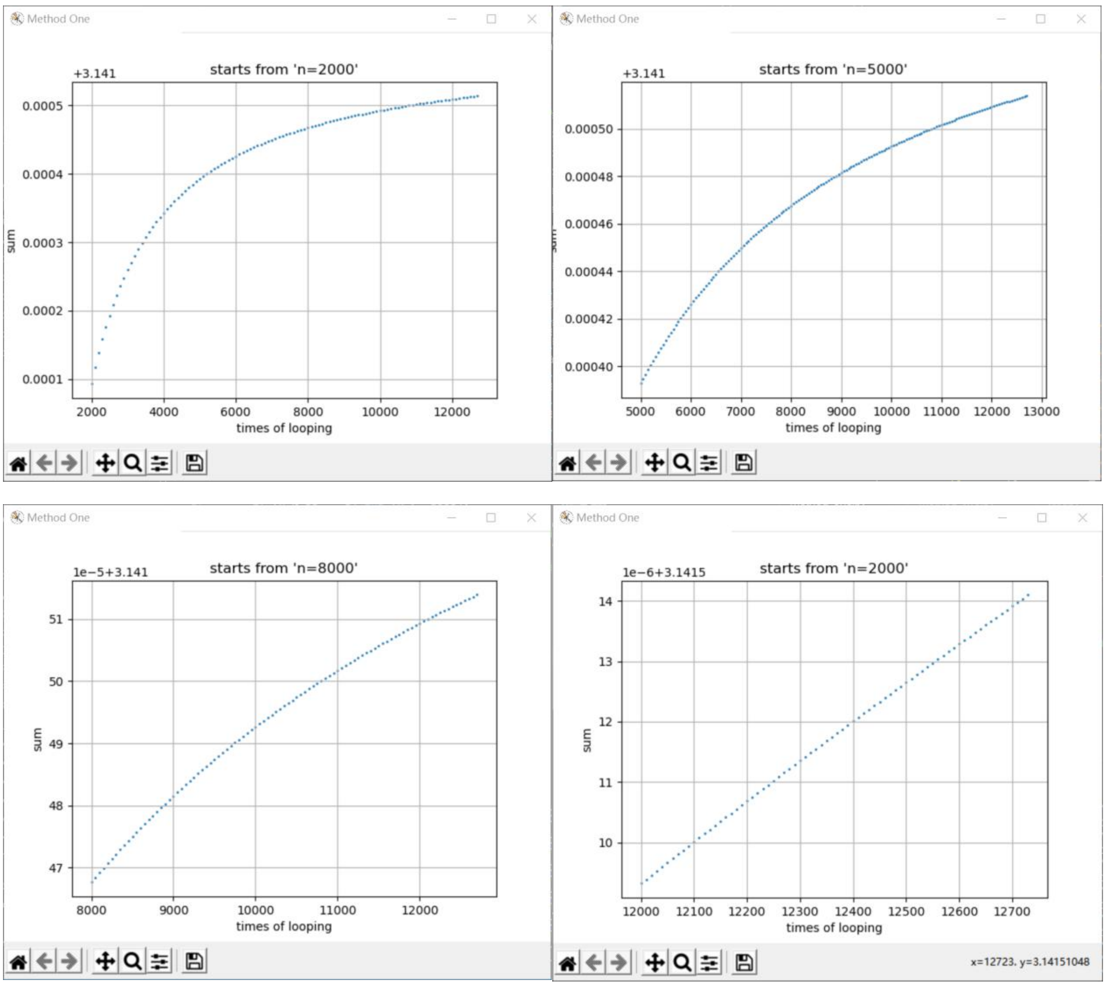
```
n=23    sum=3.141593e+00    error_a=1.442731e-16
```

五、进一步分析

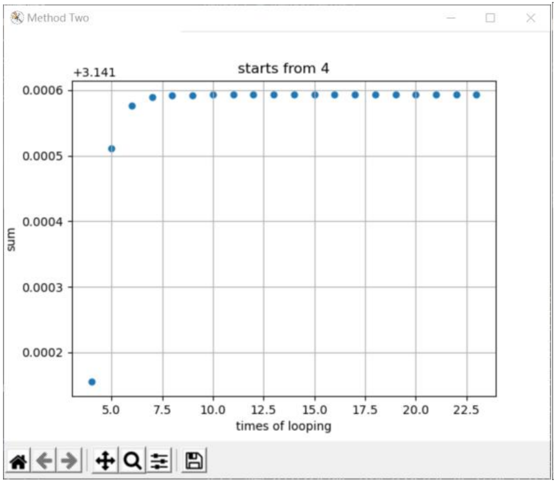
两种计算π的方法运行效率相差甚远，我们来分析一下第一种方法效率低的原因，将两种方法可视化（使用 python 实现，代码另附）：

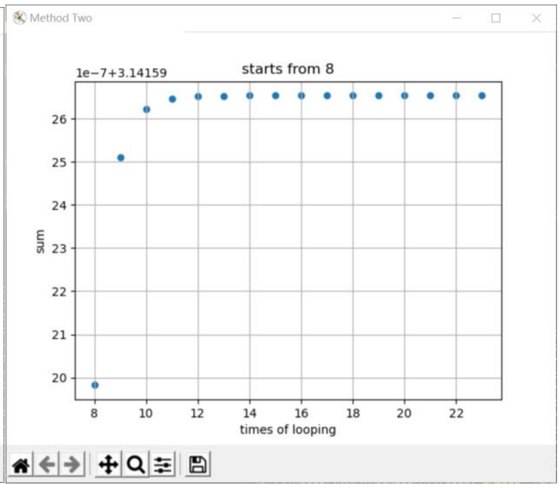


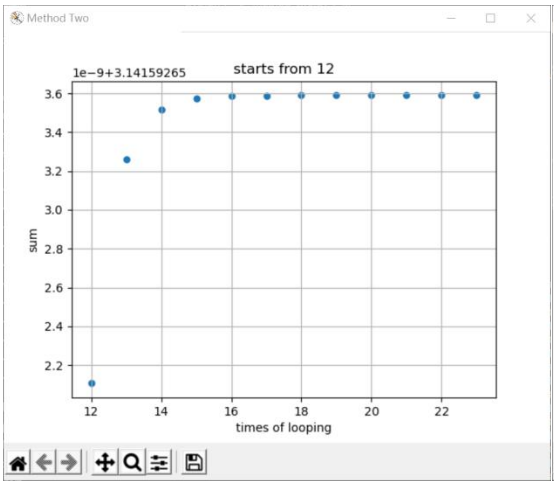
然后对第一种方法的后半段进行放大，分别从 n=2000、5000、8000、12000 处将其可视化：

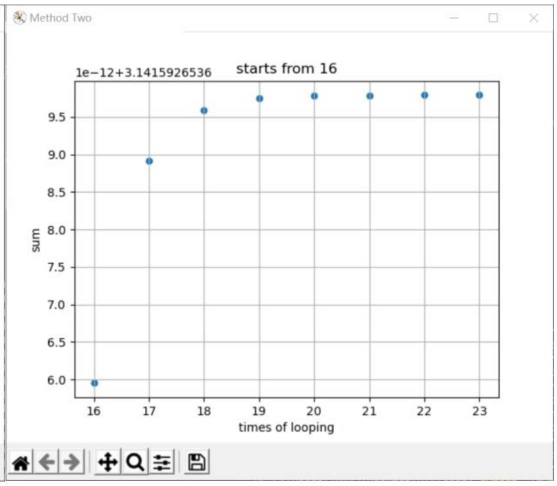


再以双精度机器精度为容限计算第二种方法，分别从 n=4、8、12、16 处将其可视化：









可以看到，随着项数的增加，第一种方法的和值的一阶导趋于不变，这是因为交错级数的正负项互相抵消后产生的和相对固定，但又由于每一项本身的值较大，因此始终不能达到循环的终止条件，导致最终求得的值比真值偏大。

而第二种方法的自身收敛速率非常快，仅仅运行 6 次就达到了容限精度，改用双精度的机器精度作为容限后运行了 23 次，运行次数仍远远小于第一种方法，且精度更高。并且可以发现，无论从哪一项开始放大，其后续的和值变化趋势都几乎一致，也就意味着数列本身的收敛性非常强，因此尽管精度再高，每一项都能迅速趋向于零达到循环终止条件，且总和与真值十分接近。

所以在采用多项式逼近某一个值时，要尽量避免交错级数的方法，一是由于可能会产生拖尾效应，导致后续大数吃掉之前的小数（本题未发生），二是由于交错级数的收敛性不强，使得运行效率大大降低，而每一项的值又相对较大，导致最后的结果精度较低。