Chap1 数值计算导论 实验报告

李晨昊 2017011466

运行代码方法

环境要求：rust，gnuplot

运行代码

```
cargo run
```

q1.1

任务

实现例1.4，体会两种误差对结果的不同影响

解题思路

仿照课本上例1.4的图形的取值范围，在对数坐标系下均匀取点，分别计算截断误差，舍入误差，总误差限，实际误差。其中实际误差表示为

$$real(x) = abs((sin(1+x) - sin(1))/x - cos(1))$$

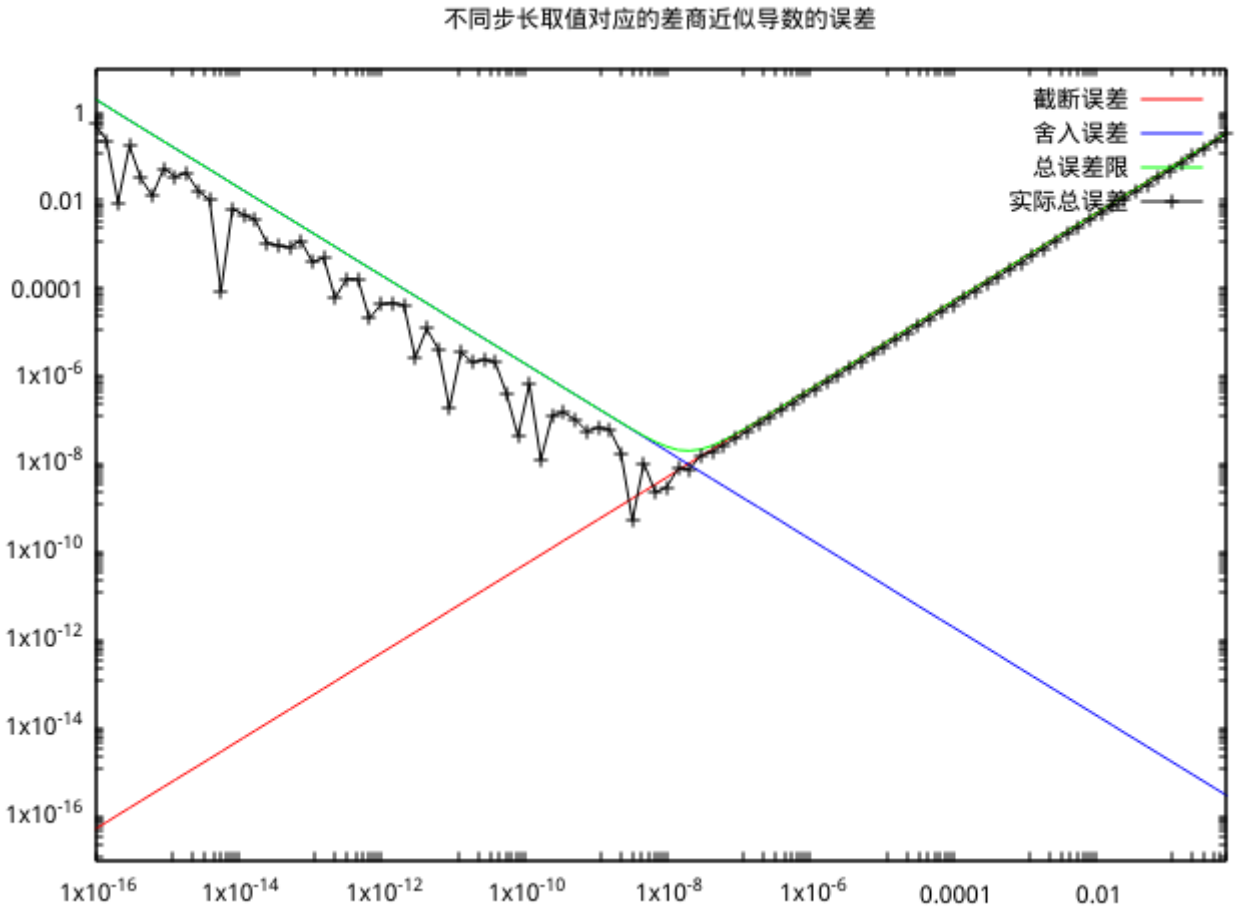
之所以加上绝对值函数，是因为实际误差存在正负，但是负数没办法在对数坐标系下表示出，所以统一取绝对值。

实际操作时，在10^{-16}到1之间的对数坐标系下均匀取了100个点，做出的图形比较清晰，也能准确反映误差变化的趋势。

实验结果

成功重现了课本例1.4的结果，即截断误差随步长的增大而增大，舍入误差随步长的增大而减小，总误差限和实际误差都先减小后增大，二者图形走势基本符合。

图形如下：



心得体会

这个知识点是我以前未曾了解的，按照我以前的想法，肯定是步长越短对近似越有利。现在通过课本的知识了解到并不是这样的。通过这次实验我更加加深了对这个知识的理解。

我认为这个规律应该也可以应用到别的与科学计算相关的领域，例如粒子运动的模拟等，也应该合适地选择步长，而不是越小越好。

q1.3

任务

编程观察调和级数的求和计算

1. 求出采用IEEE单精度浮点数计算时，n为多少时结果不再变化
2. 在1中的n下，使用IEEE双精度浮点数计算，比较1和2的结果以评估IEEE单精度浮点数的误差
3. 估计采用IEEE单精度浮点数计算时，n为多少时结果不再变化

解题思路

rust标准库中提供了

```
const EPSILON: f64 = 2.2204460492503131e-16_f64;
```

按照文档的描述，这是课上提到的ϵ_{mach}的两倍，于是将其除以二以得到ϵ_{mach}。输出它与课本上的结果比较，结果是符合的。

第1，2两个小问直接计算即可。

第3小问，由于"大数吃掉小数"发生在$small/big < \epsilon_{mach}$时，经过一定的估计，本题的$n$大致在$10^{14}$级别，直接计算比较耗时，因此使用$ln(n)+\gamma$来估计$\Sigma_{i=1}^{n}(\frac{1}{i})$。

具体做法是，先将n不断加倍，直到$1/(ln(n)+\gamma)/n < \epsilon_{mach}$，再在$\frac{n}{2}$和$n$间进行二分，将$n$精确到个位。当然，我明白第二步的意义其实不大，因为实际上求和时误差的积累会很容易对结果产生影响，因此这样的估计应该只有数量级是有意义的，精确到个位则没有什么意义。

实验结果

输出为

```
对于IEEE单精度浮点数，n=2097152时结果不再变化

估计IEEE单精度浮点数计算结果的误差为0.2703764904991992

如采用IEEE双精度浮点数，估计在n=266534544114701时求和结果不再变化
按照普通计算机每秒进行10^9次运算来估算，这需要266534s，即3.085天
```

心得体会

在实际的编程当中，我使用单精度浮点数比较多；在oj上做题时，我使用双精度浮点数比较多，这基本上就是个人习惯，我很难说清楚其中的道理来。因此，做实验前，我没有预料到单精度浮点数和双精度浮点数之间的差距会有这么大。

本次实验帮助我加深了对单精度浮点数和双精度浮点数精度差距的理解(但是对于速度差距，则没有提供什么新的知识)，我以后会更加仔细的考虑浮点数的应用场景，选择合适的浮点数类型。