数值计算导论 实验报告

计73 郑林楷 2017011474

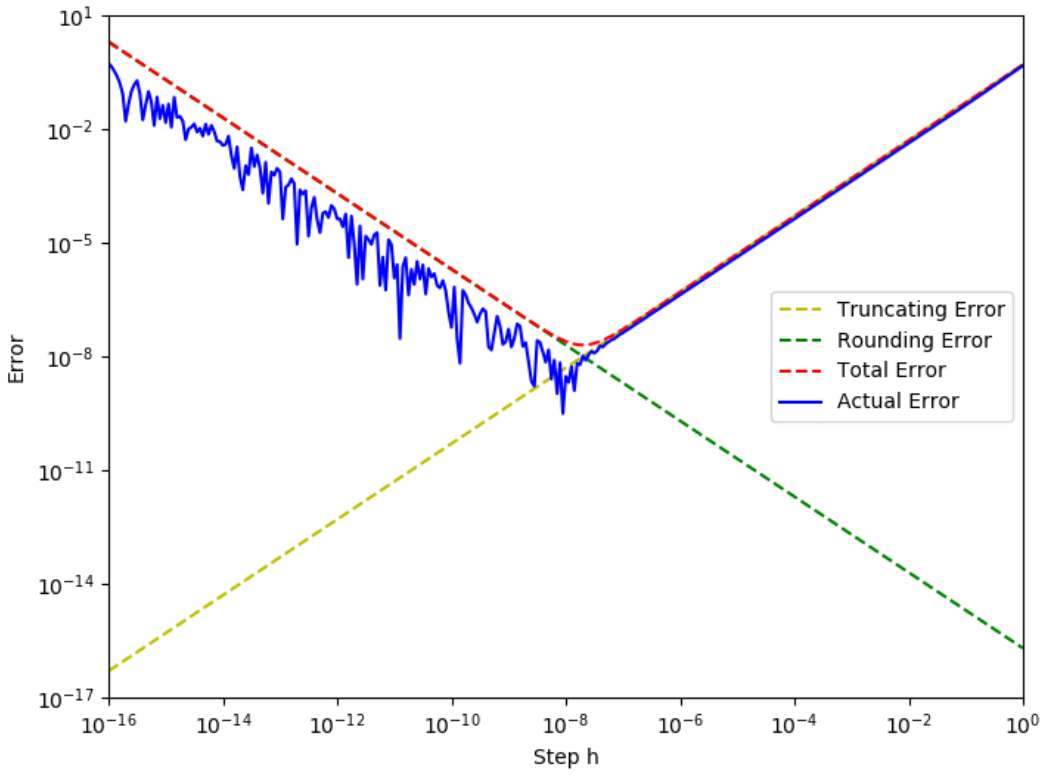
我使用 python3 + numpy + scipy + matplotlib 完成以下若干实验。

1-1

代码可见于 `lab1/1-1.py`。

运行结果

采用对数坐标轴画图。



实现过程

设定 ϵ 为 2^{-24}，步长 h 在对数空间上面线性取点。

画图的时候，只需要画出 $\frac{Mh}{2}$、$\frac{2\epsilon}{h}$、$\frac{Mh}{2}+\frac{2\epsilon}{h}$ 和 $|\frac{\sin(x_0+h)-\sin(x_0)}{h}-\sin'(x_0)|$，其中 $x_0=1$，分别对应图中的 `Truncation Error`、`Rounding Error`、`Total Error` 和 `Actual Error`。

结果分析和实验总结

可以看到截断误差随 h 的增加而增加，舍入误差随 h 的增加而减少，实际误差在 $h=10^{-8}$ 的时候最小，实际使用商差法时应该选取这一数值，使得误差最小。

1-3

代码可见于 `lab1/1-3.py`。

运行结果

```
Stop at n = 2097152, sum = 15.403682708740234
Theoretical value: n = 2195967
Absolute error = 0.270376013662041
Relative error = 1.787%
With float64, n will be 578556828663210
Will stop in 4.018 hours
```

实现过程

根据理论分析，我们可以知道当 $\frac{1}{n} \leq \frac{1}{2}\epsilon_{\text{match}} \sum_{k-1}^{n-1} \frac{1}{k}$ 时结果在之后的运算中将不会发生变化。

```python
n = 1
sum = np.float64(0)
eps = 6e-8

while True:
    tmp = np.float64(np.float64(1) / np.float64(n))
    if tmp <= eps * sum / 2:
        break
    sum += tmp
    n += 1
print(f'Theoretical value: n = {n}')
```

调和级数近似为 $\ln n + \gamma + \frac{1}{2n}$（$\gamma$ 为欧拉常数）。如果使用双精度浮点数进行计算，则将上文中作为终止条件的不等式转化为 $\frac{1}{n} \leq \frac{1}{2}\epsilon_{match}(\ln n + \gamma + \frac{1}{2n})$。通过测试得知当前笔记本的浮点计算性能峰值为 80 GFLOPS。

```python
from scipy.optimize import fsolve

def f(n):
    gamma = 0.57721566490153286060651209008240243104215933593992
    return 1e-16 * (np.log(n) + gamma + 1 / (2 * n)) / 2 - 1 / n

n = int(fsolve(f, [1.])[0])
print(f'With float64, n will be {n}')
print(f'Will stop in {n * 2 / (80 * 1e9) / 3600:.3f} hours')
```

结果分析和实验总结

最后得到的结果「本机上计算需要 4.018 小时」只是基于理论计算，实际情况因为 CPU 性能表现不稳定等硬件因素以及非零开销抽象语言（例如 Python）自身过大的常数，都可能会让实际运行时间远比这个数值要大。

通过这次实验，我真实感受到了浮点数在不断运算积累之后所带来的误差是如此的大，所以在日常计算的时候，如果涉及到小数，要特别注意进度所带来的误差问题。