《数值分析》课程实验报告

实验名称___实验 1　数值计算的基本概念___________

班级		**姓名**		**学号**		**序号**	
教师		**地点**	数学实验中心			**评分**	

一、　实验目的

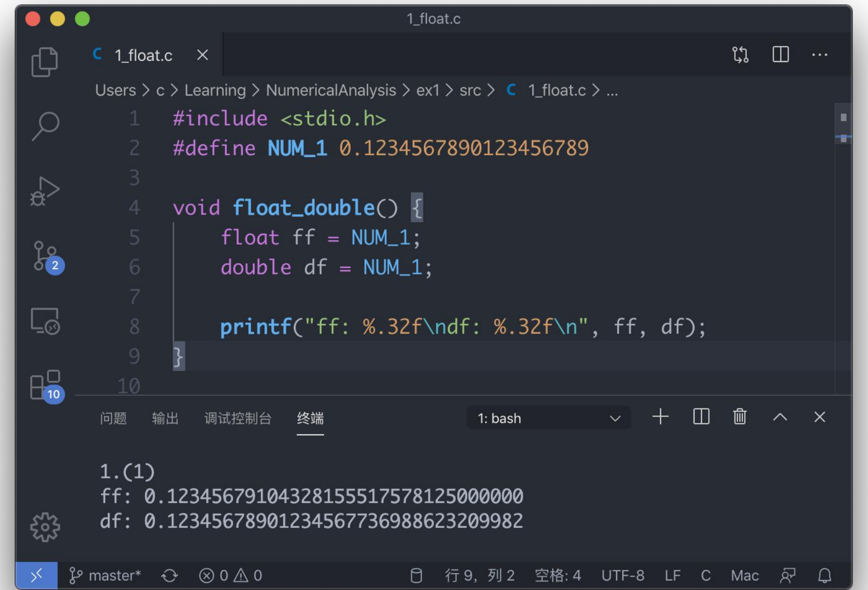
① 了解计算机中浮点数的有效数字；
② 了解舍入误差产生的原因，知道截断误差和舍入误差的区别；
③ 了解算法“稳定性”的概念；
④ 了解“病态问题”的概念。

二、实验过程和结果

1、关于浮点数

（1）令 $ff = 0.12345678901234567890$; $df = 0.12345678901234567890$;

在计算机中分别将它们定义成单精度型和双精度型，输出观察结果，并对结果进行分析。



（程序详见 https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/1_float.c）

从上图程序即运行结果截图中，可以看到 float 变量有 7 位有效数字，double 类型有 16 位有效数字。

（2） 设 $a1 = 1.000001; a2 = 1.000000; a3 = 0.0000001$， 在单精度的变量环境下做以下操作：

1） 按以下两种算法计算 $a1$ 与 100 个 $a3$ 相加的结果。

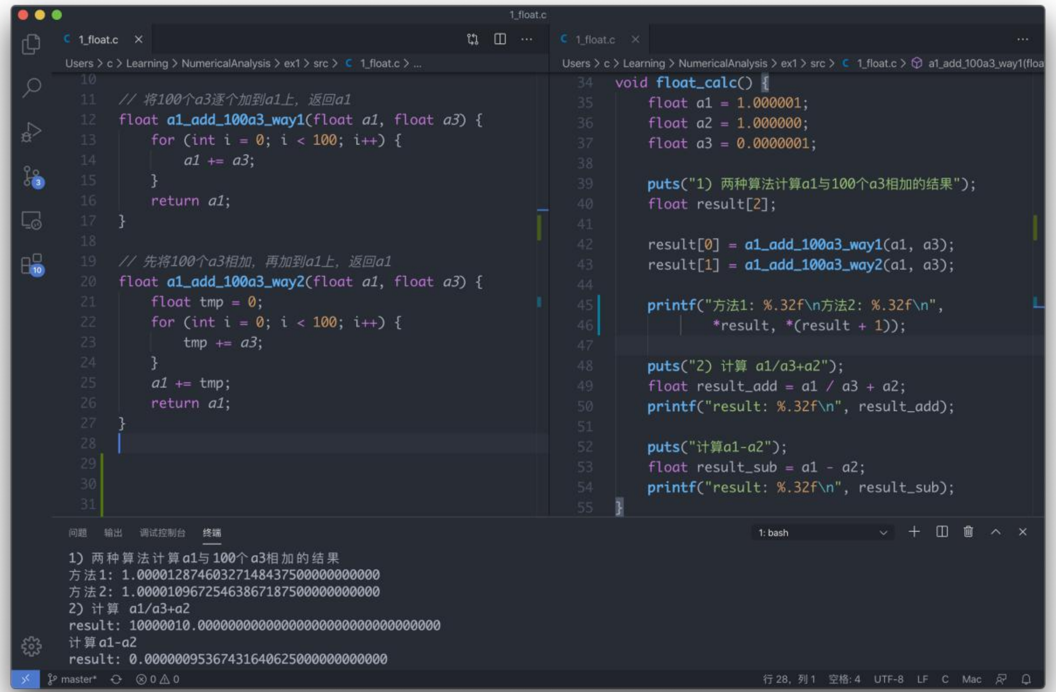
方法一：将 100 个 $a3$ 逐个加到 $a1$ 上；

方法二：先将 100 个 $a3$ 相加，再加到 $a1$ 上；

观察所得到的结果，写出你得到的结论。

2） 计算 $\dfrac{a1}{a3} + a2$，观察结果，并分析原因。

3） 计算 $a1-a2$,观察有效数字的位数，从中你可以得到什么启示?



（程序详见 https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/1_float.c）

可以看到浮点数的运算并不精确，不同的运算顺序会得到有所区别的结果。较小的数加到较大的数上会造成较小的值丢失；较小的数做分母、相近的数相减，都会损失有效数字。

在该题实验中，使用不同编译器优化等级进行编译，可能造成结果的差异，详见本人的文章 https://blog.csdn.net/u012419550/article/details/109137101

2、舍入误差

考虑计算一元可微函数 $f(x)$在 x_0 处导数的近似方法,

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h} \quad (1)$$

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0-h)}{2h} \quad (2)$$

取 $f(x)=x^3$，分别用（1）、（2）计算 $f(x)$在 $x_0=1$ 处的一阶导数 $f'(x_0)$ 的近似值，令 h 依次取值 $1,10^{-1},10^{-2},\cdots,10^{-15}$，观察所得结果并与精确值进行比较，结合本例叙述你对于截断误差和舍入误差的认识。

```
double df_1(double (*f)(double), double x0, double h) {
    return (f(x0 + h) - f(x0)) / h;
}

double df_2(double (*f)(double), double x0, double h) {
    return (f(x0 + h) - f(x0 - h)) / (2 * h);
}

double f(double x) { return x * x * x; }

int main() {
    double x0 = 1;
    double h = 1;
    for (int i = 0; i < 16; i++) {
        double diff_result[2];  // 存放 df_1、df_2 的结果
        diff_result[0] = df_1(f, x0, h);
        diff_result[1] = df_2(f, x0, h);
        h /= 10;
    }
    return 0;
}
```

(此处省略了部分 IO 相关代码，完整代码见：
https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/2_round_off_error.c)

运行结果：

```
--- h = 1 ---
df_1: 7.000000
df_2: 4.000000

--- h = 0.1 ---
df_1: 3.310000
df_2: 3.010000

--- h = 0.01 ---
df_1: 3.030100
df_2: 3.000100

--- h = 0.001 ---
df_1: 3.003001
df_2: 3.000001

--- h = 0.0001 --
df_1: 3.000300
df_2: 3.000000

--- h = 1e-05 ---
df_1: 3.000030
df_2: 3.000000

--- h = 1e-06 ---
df_1: 3.000003
df_2: 3.000000

--- h = 1e-07 ---
df_1: 3.000000
df_2: 3.000000

--- h = 1e-08 ---
df_1: 3.000000
df_2: 3.000000

--- h = 1e-09 ---
df_1: 3.000000
df_2: 3.000000

--- h = 1e-10 ---
df_1: 3.000000
df_2: 3.000000

--- h = 1e-11 ---
df_1: 3.000000
df_2: 3.000000

--- h = 1e-12 ---
df_1: 3.000267
df_2: 3.000100

--- h = 1e-13 ---
df_1: 2.997602
df_2: 2.999268

--- h = 1e-14 ---
df_1: 2.997602
df_2: 2.997602

--- h = 1e-15 ---
df_1: 3.330669
df_2: 3.164136
```

舍入误差是由计算机的有限精度所引起，截断误差是算法中对于原问题的近似引起。截断误差的问题可以通过改进算法来改善。

3、算法的稳定性

考虑积分

$$I_n = \int_0^1 \frac{x^n}{x+5} dx,$$

易见,

$$I_n + 5I_{n-1} = \int_0^1 \frac{x^n + 5x^{n-1}}{x+5} dx = \int_0^1 x^{n-1} dx = \frac{1}{n},$$

且计算得到 $I_0 = \ln\frac{6}{5} = 0.1823$,

从而可得如下递推算法：

$$I_n = \frac{1}{n} - 5I_{n-1} \ ,(n = 1,2,3,\cdots,100) \qquad (1)$$

对上述积分有估计式

$$\frac{1}{6(n+1)} < I_n < \frac{1}{5(n+1)},$$

我们取 $I_{100} \approx \frac{1}{2}(\frac{1}{606} + \frac{1}{505}) \approx 0.001815$ ，可得另一个递推算法：

$$I_{n-1} = \frac{1}{5}(\frac{1}{n} - I_n) \ , (n = 100, 99, \cdots, 2, 1) \qquad (2)$$

（已知 $I_8 = 0.01884, I_{10} = 0.01536, I_{12} = 0.01297, I_{14} = 0.01123$）

分析算法（1）和（2），哪一个算法稳定，并编程验证你的结论!

```
#define I0 0.1823
#define I100 0.001815
#define TEST_COUNT 4

double algo_1(int n) {
    if (n == 0) {
        return I0;
    }
    return 1.0 / n - 5.0 * algo_1(n - 1);
}

double algo_2(int n) {
    if (n == 100) {
        return I100;
    }
    return (1.0 / n - algo_2(n + 1)) / 5.0;
}
```

(完整代码见：
https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/3_algo_stability.c)

运行结果：

```
algo_1: I(n) = 1 / n - 5 * I(n - 1)
algo_2: I(n) = (1 / n - I(n + 1)) / 5

accuracy: I(8) = 0.018840
algo_1: -8.401786
algo_2: 0.021233

accuracy: I(10) = 0.015360
algo_1: -210.500198
algo_2: 0.016926

accuracy: I(12) = 0.012970
algo_1: -5262.876172
algo_2: 0.014071

accuracy: I(14) = 0.011230
algo_1: -131572.217498
algo_2: 0.012040
```

可以看到算法 1 较算法 2 更不稳定。

4、病态问题

考虑二阶线性方程组

$$\begin{cases} x_1 + ax_2 = 1 \\ ax_1 + x_2 = 0 \end{cases} \tag{1}$$

$$\begin{cases} x_1 + ax_2 = 1 \\ ax_1 + 4x_2 = 0 \end{cases} \tag{2}$$

令 a 分别取 0.99 和 0.991，求解计算上述方程组。

对（1）式：

$$\begin{cases} x_1 + ax_2 = 1 \\ ax_1 + x_2 = 0 \end{cases}$$

```
e = [
    x1 + a * x2 - 1,
    a * x1 + x2,
]
```

```
r = solve(e, [x1, x2])

for n in [0.99, 0.991]:
    print(f"a={n}")
    for k in r:
        print(f"{k}: {r[k].subs(a, n)}")
    print("---")
```

```
a=0.99
x2: -49.7487437185929
x1: 50.2512562814070
---
a=0.991
x2: -55.3044254701713
x1: 55.8066856409397
---
```

对（2）式：

$$\begin{cases} x_1 + ax_2 = 1 \\ ax_1 + 4x_2 = 0 \end{cases}$$

```
e = [
    x1 + a * x2 - 1,
    a * x1 + 4 * x2,
]

r = solve(e, [x1, x2])

for n in [0.99, 0.991]:
    print(f"a={n}")
    for k in r:
        print(f"{k}: {r[k].subs(a, n)}")
    print("---")
```

```
a=0.99
x2: -0.327825424682937
x1: 1.32454717043611
---
a=0.991
x2: -0.328371967571032
x1: 1.32541661986289
---
```

（完整程序：
https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/4_pathological.ipynb）

结果表现出，（1）式在 a 变化很小时即出现了很大的结果差异。也就是说这个方程对误差较为敏感，存在病态问题。

三、思考题分析解答

1、简述什么是数值稳定和数值不稳定?

舍入误差在计算过程中呈现衰减态势的算法称是稳定的，否则称之为不稳定的。往往稳定的算法才是有用的。

2、什么是病态问题?

病态问题是微小的误差就会引起目标值的很大变化的问题。不病态的问题称为良态问题。常规方法对于病态问题往往是失效的。

3、运用如下迭代公式计算 $\sqrt{a}$ （$a>0$），初值 $x_0>0$ 为 $\sqrt{a}$ 的某一近似值，并且要求当 $\left|\frac{x_k-x_{k-1}}{x_k}\right|<10^{-5}$ 满足时停止迭代，并输出结果！取不同的初值，观察迭代次数的变化，并记录。

$$x_{k+1}=\frac{1}{2}\left(x_k+\frac{a}{x_k}\right),\quad k=0,1,2,\cdots$$

关键代码:

```
// iter 从 x_0 = x0 开始，做 x_{k+1} = f(x_k, a) 的迭代
// 直到 abs((x_k - x_{k-1}) / x_k) < tol
double iter(double (*f)(double xk, double a), double a, double x0, double tol)
{
    double xk_prev;
    double xk = x0;
    int count = 0;
    do {
        xk_prev = xk;
        xk = f(xk, a);
        count++;
    } while (absd((xk - xk_prev) / xk) >= tol);
    return xk;
}
// 迭代函数
double f(double xk, double a) { return (xk + a / xk) / 2.0; }
```

（省略了部分代码，完整代码见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex1/src/think_3.c）

运行结果：

```
a: 9.000000       x0: 1.000000     iter_times: 6   result: 3.000000
a: 9.000000       x0: 2.000000     iter_times: 4   result: 3.000000
a: 9.000000       x0: 2.500000     iter_times: 4   result: 3.000000
a: 9.000000       x0: 3.000000     iter_times: 1   result: 3.000000
a: 9.000000       x0: 3.500000     iter_times: 4   result: 3.000000
a: 9.000000       x0: 4.000000     iter_times: 4   result: 3.000000
a: 9.000000       x0: 100.000000  iter_times: 9   result: 3.000000
```

四、重点难点分析

重点：

1. 了解计算机中浮点数的有效数字；
2. 了解舍入误差产生的原因，知道截断误差和舍入误差的区别；
3. 了解算法“稳定性”的概念；
4. 了解“病态问题”的概念。

难点：

1. 在实验浮点数的有效数字时注意输出位数的控制。
2. 在舍入误差、思考题 3 中，为了做通用的表达，让程序根据清晰、可复用，使用基本的函数指针技术。

注：本实验中全部 C 程序均在 macOS Catalina 10.15.6 下使用 Homebrew GCC 9.2.0 编译测试通过。另外部分代码使用 Python 3.7.6 在 JupyterLab 2.2.8 中完成。有关代码可以从 https://github.com/cdfmlr/NumericalAnalysis/tree/master/ex1/src 获取。