《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** |  |
| 1. 实验目的 2. 了解计算机中浮点数的有效数字； 3. 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； 4. 了解算法“稳定性”的概念； 5. 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  截屏2020-10-30 19.52.14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，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*a*1与100个*a*3相加的结果。  方法一：将100个*a*3逐个加到*a*1上；  方法二：先将100个*a*3相加，再加到*a*1上；  观察所得到的结果，写出你得到的结论。  2）计算，观察结果，并分析原因。  3）计算*a*1-*a*2,观察有效数字的位数，从中你可以得到什么启示？  **截屏2020-10-30 19.58.37**  （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，  (1）  （2）  取，分别用（1）、（2）计算*f*(*x*)在处的一阶导数的近似值，令*h*依次取值，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：  截屏2020-10-30 20.25.20  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    易见，  ，  且计算得到，  从而可得如下递推算法：  （1）  对上述积分有估计式  ，  我们取，可得另一个递推算法：  （2）  （已知 ）  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：  截屏2020-10-30 20.38.13  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  （1）  （2）  令*a*分别取0.99和0.991，求解计算上述方程组。  对（1）式：  截屏2020-10-30 20.43.02  对（2）式：  截屏2020-10-30 20.44.45  （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。   1. 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（），初值为的某一近似值，并且要求当满足时停止迭代，并输出结果！取不同的初值，观察迭代次数的变化，并记录。    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：  截屏2020-10-30 20.58.21   1. 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。 | | | | | | | |