山东大学 计算机 学院

数值计算 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201300140122 | 姓名： 于仲源 | | 班级：2013级2班 |
| 实验题目： (1)无穷级数(2)一元二次方程的求解 | | | |
| 实验学时：4 | | 实验日期： 2017/3/15 | |
| 实验目的：  认识数值计算中浮点系统和浮点运算的特点 | | | |
| 硬件环境：  Thinkpad 笔记本 | | | |
| 软件环境：  VS2013 | | | |
| 实验步骤与内容：  (1)无穷级数  (a) 证明级数是发散的：    (b) 为什么浮点运算中级数的和是有限的？  精度原因，浮点系统中表示的数的范围是有限的。当级数的第k项过小以致浮点数的精度不够计算为0，所以后面的项被忽略掉，和有限。  (c) 在ieee单精度和双精度浮点运算中，什么时候部分和停止变化？若已知计算机浮点操作的执行速度，上述计算需要执行多长时间？  当满足    时，部分和停止变化。使用单精度，经计算机计算n=566153时，和为14.8168时停止运算。实际执行时间近似为566153\*浮点操作时间。使用双精度时，停止时间较长。  (d) 编写两个计算级数和的程序，一个是单精度另一个是双精度，使用什么判停准则？实际的结果怎样？实际结果与预期结果比较，以及运行时间？  判停准则是项的相对数值小于浮点数精度。单精度时实际结果只运行到数字十几，理论上是无穷大（级数发散）。实验结果符合预期，毕竟在计算机上运行存在由浮点系统产生的偏差。双精度时，实验结果相对令人满意，运行时间很长，计算结果的精度比使用单精度表示数据时高许多。  单精度程序：  // @yzy  //  #include<stdio.h>  #include<math.h>  #include<float.h>  void jishu(){  float n = 1;  float s = 1;  while (1 / n >= FLT\_EPSILON\*s){  s += 1 / n;  n++;  }  printf("n 的值是 %f\n", n);  printf("s 的值是 %f\n", s);  }  int main(){  jishu();  return 0;  }  双精度程序：  // @yzy  //  #include<stdio.h>  #include<math.h>  #include<float.h>  void jishu(){  double n = 1;  double s = 1;  while (1 / n >= DBL\_EPSILON\*s){  s += 1 / n;  n++;  if (n == 100){  printf("n 的值是 %f\n", n);  printf("s 的值是 %f\n", s);  printf("==============\n");  }  if (n == 1000){  printf("n 的值是 %f\n", n);  printf("s 的值是 %f\n", s);  printf("==============\n");  }  if (n == 10000){  printf("n 的值是 %f\n", n);  printf("s 的值是 %f\n", s);  printf("==============\n");  }  if (n > 10000000)break;    }  printf("n 的值是 %f\n", n);  printf("s 的值是 %f\n", s);  }  int main(){  jishu();  return 0;  }  单精度结果：  输出n与s的数值：    双精度结果：  程序修改前：输出n与s的数值：（很难将程序运行到底）    修改程序后：    程序运行过程中n很快就达到了10000000（远远超过单精度表示时相应的数值）  (2)一元二次方程的求解  程序：  // @yzy  //  #include<stdio.h>  #include<math.h>  #define DBL\_MAX 1.7976931348623158e+308 /\* max value \*/  int SQuadratic(){  //printf("Hello world!\n");  printf("flt\_max %Ef\n", DBL\_MAX);  double x1;//x1,x2分别为方程的2个解  double x2;  double melt;  double a;  double b;//ABC的三个变量  double c;  printf("请输入a b c三个数的值：\n");  //scanf("%lf %lf %lf", &a, &b, &c);//输入ABC的时候需要空格  a = 10E-154;  b = -10E154;  c = 10E154;  //输入检测  if (a == 0|| c==0 ){  printf("输入有误！\n");  return -1;  }  //判断是否溢出  if (abs(b) >= sqrt(DBL\_MAX)){  //printf("%ef", b);  double temp\_b = 1.0 ;  double temp\_c = c / b;  double temp\_a = a / b;  melt = 1.0 - 4 \* temp\_c\* temp\_a;  printf("%ef", melt);  if (melt >= 0){  x1 = (-temp\_b + sqrt(melt)) / 2;  x2 = (-temp\_b - sqrt(melt)) / 2;  printf("溢出，2个解,x1 = %lf,x2 = %lf\n", x1, x2);  }  else{  double d = (double)-b / 2;  double e = sqrt(-melt) / 2;  printf("存在复数解, x1的实部为%lf, x1的虚部为%lf . x2的实部为%lf, 虚部为%lf \n", d, e, d, -e);  }  }  //如果没有溢出  else{  melt = b\*b - 4 \* a\*c;//初始化melt的值  //printf("%lf", melt);  if (melt>0){  //针对抵消问题，x1使用传统公式，x2使用变形后的公式  if (b<0){  x1 = (-b + sqrt(melt)) / (2 \* a);  x2 = (2 \* c) / (-b + sqrt(melt));  }  //针对抵消问题，x2使用传统公式，x1使用变形后的公式  else {  x1 = (2 \* c) / (-b - sqrt(melt));  x2 = (-b - sqrt(melt)) / (2 \* a);  }  printf("2个解,x1 = %lf,x2 = %lf\n", x1, x2);  }  else if (melt == 0)  {  x1 = (-b) / (2 \* a);  x2 = x1;  printf("1个解,x1 = %lf,x2 = lf%\n", x1, x2);  }  else  {  double d = (double)-b / (2 \* a);  double e = sqrt(-melt) / (2 \* a);  printf("存在复数解, x1的实部为%lf, x1的虚部为%lf . x2的实部为%lf, 虚部为%lf \n", d, e, d, -e);  }  }  return 0;  }  int main(){  SQuadratic();  return 0;  }  实验运行结果：  a b c res1 res2  6 5 -4 0.500000 -1.333333  6\*10E154 5\*10E154 -4\*10E154 0.500000 -1.333333  0 1 1 输入有误，a=0  1 -4 3.999999 1.999000 2.001000  10E-155 -10E155 10E155 0.000000 -1.000000  实验截图： | | | |
| 结论分析与体会：  本次实验是一个很好的锻炼机会，将书中的公式付诸实践。这让我更加深刻的理解了公式的内涵与运用的方式。  实验中二次方程的求解，是一个很好的分支程序的例子，要求我们思维严谨。以及对于抵消问题的处理，让我觉得自己确实学到好多东西。 | | | |