数值计算第一周上机实验报告

211240021 田铭扬 2023-2-26

·1 问题

编程找出自己的计算机 float 类型与 double 类型的机器精度、下溢值和上溢值。

·2 原理与算法思路

·2.1“位”形式与浮点数形式的转换

由于计算机存储 float 类型和 double 类型的方式比较复杂，为了更加直观，我需要一个程序，按位显示一个 float 或 double 类型浮点数在内存中的“位形式”。同时我们需要一个反向程序，使得我能够输入一串二进制位，让它转化为一个浮点数。

实现方法很简单。以 float 为例，强制让 unsigned int*类型的指针 pui 与 float*类型指针的 pf 指向同一地址。那么正向地，只需用 pf 读入一个浮点数，然后对 pui 用二进制输出即可；反之亦然，读入二进制数，储存到 pui 中，再用 pf 输出。代码如下：

注：此处为 float 的代码，double 只需相应修改变量类型即可。

```
#include <cstdio> //float to bit
using namespace std;
int main() {
  float *pf = 0;
  unsigned int *pui = 0;
  pf=new(float);
  while(1){
    scanf("%f",pf);
    pui = (unsigned int *)pf;
    for(int i=31;i>-1;i--)
      printf("%d",((*pui)>>i)%2);
    printf("\n\n");
  }
  return 0;
}
```

```
#include <cstdio> //bit to float
using namespace std;
int main() {
  float *pf = 0;
  unsigned int *pui = 0;
  pui=new(unsigned int);
  char input_c;
  while(1){
    *pui=0;
    for(int i=31;i>-1;i--){
      scanf("%c",&input_c);
      while(input_c<'0'||input_c>'9')
        scanf("%c",&input_c);
      *pui+=(input_c-'0')<<i;
    }
    pf = (float *)pui;
    printf("%.50f\n\n",*pf);
  }
  return 0;
}
```

·2.2 机器精度

由定义知，设相邻的两个机器数分别为 a 和 b，则在区间[a,b]中相对误差最大（精度最低）一点为(a+b)/2，相对误差为(a-b)/(a+b)。而由计算机存储 float 类型的方式知，[a,b]与[a*2^k,b*2^k](k 为整数)相应的精度是一致的[*]，故只需遍历[1,2]间的所有相邻机器数，计算(a-b)/(a+b)的最大值即可。代码如下：

```
#include <cstdio> //float
using namespace std;
float a,b;
double max_ep=-1.0;
double epsilon(double a, double b){ return (a-b)/(a+b); }
int main() {
   unsigned int *pui = new(unsigned int);
   float *pf = (float *)pui;
   *pf = 1.0;
   for(int i=0;i<4194304;i++){ //2^22
     a=*pf; *pui+=1; b=*pf;
     max_ep=max(max_ep,epsilon(b,a));
   }
   printf("%.30f",max_ep);
   return 0;
}
```

·2.3 上溢值与下溢值

上溢值和下溢值可以使用 2.1 中的程序“半自动”地得出。

对于 float 类型的上溢值：通过查询资料知，其“位形式”为 01111111 01111111 11111111 11111111（指数位第 9 位为’0’，因为若为’1’，则值为 nan）, 对应的值为 340,282,346,638,528,859,811,704,183,484,516,925,440（十进制）或 ffff ff00 0000 0000 0000 0000 0000 0000（十六进制），记为 X。若要验证这一结论的正确性，只需取 X 的上一个机器数 Y（事实上，Y 为 01111111 01111111 11111111 11111110），然后写一个 float 类型的“A+B”程序，计算 X+(X-Y)或者 X+(X-Y)*2。若前者为 inf，或前者等于 X 而后者为 inf，则 X 就是上溢值。

对于下溢值：查资料得知 其“位形式”为 00000000 00000000 00000000 00000001（即指数部分为 000000000，查资料知此时存储的为非规格化的浮点数），即二进制的 2^(-149)——事实上，149=126+23——约等于十进制的 1.40*10^(-45)。验证方法与上溢值类似，记 X=1*2^(-149)，Y 为其前一个机器数（事实上，Y 为 00000000 00000000 00000000 00000010），计算 X-(Y-X).若结果为 0，则 X 就是下溢值。

对于 double 类型，是同理的。

·3 实验过程与结果

·3.1 机器精度

运行 2.2 中的代码即可得到 float 类型的机器精度，得到二进制的 2^(-24)，约等于十进制的 5.96*10^(-8)。

由于运算的速度限制，double 类型无法使用 float 类型的代码进行改动。但是注意到，沿用 2.2 中的记号[a,b], 在 a=1 时相对误差最大，为 2^(-24)；相对误差随 a 变大而变小，至 b=2 时误差为 2^(-25)最小；而 a=2 时，误差又变回 2^(-24)（这恰好验证了 2.2 中的[*]）。因此，我们可以断言 double 类型的机器精度——即最大相对误差——也在 a=1 时取到，为二进制下的 2^(-53)，约等于的 1.11*10(-16)。

·3.2 上溢值与下溢值

实验过程如 2.3 所述，结果表明查询到的资料正确，详见“结论”部分。

·4 结论

对于我所使用的计算机，float 类型：

机器精度为 2^(-24)，约等于 5.96*10^(-8)；

上溢值为 ffff ff00 0000 0000 0000 0000 0000 0000，约等于 3.40*10^38；

下溢值为 2^(-149)，约等于 1.40*10^(-45)。

double 类型：

机器精度为 2^(-53)，约等于 1.11*10^(-16);

上溢值为 ffff ffff ffff f800*16^(240)，约等于 1.80*10^308；

下溢值为 2^(-1074)，约等于 4.94*10^(-324)。