计算方法小论文

PB16000702 韦璐

在本学期，我们进行了多次数值积分的计算，但是都是在维数比较低的空间中，现在我们试着使用 Monte Carlo 方法计算如下定积分，并讨论有效数字的位数。

$\int_0^1 dx\sqrt{x+\sqrt{x}}$,

$\int_0^{\frac{7}{10}} dx \int_0^{\frac{5}{4}} dy \int_0^{\frac{9}{10}} dz \int_0^1 du \int_0^{\frac{11}{10}} dv\,(6-x^2-y^2-z^2-u^2-v^2)$。

编程思路:

1. 我们使用蒙特卡洛简单抽样平均值法。

在计算单变量积分的时候，根据平均值定理：$\int_a^b f(x)dx = (b-a) < f >$，其中

$< \mathrm{f} > = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$，也就是<f>表示 f(x)在[a,b]上的均值，x_i在[a,b]区间中均匀随机选取。

所以有$\int_a^b f(x)dx \approx \frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i)$，计算多重积分的时候，上面的式子推广为

$$\int_{a_1}^{b_1} dx_1 \int_{a_2}^{b_2} dx_2 \ldots \int_{a_n}^{b_n} dx_n\, f(x_1, x_2, \ldots, x_n) = \frac{1}{N}\left[\prod_{j=1}^{n}(b_j - a_j)\right]\sum_{i=1}^{N} f(x_{1_i}, x_{2_i}, \ldots, x_{n_i})$$

2. 中心极限定理与误差

中心极限定理说，当 N 有限时，$\mathrm{P}\left(\left(\frac{<f>-\mu}{\frac{\sigma_f}{\sqrt{N}}}\right) < \beta\right) \to \phi(\beta)$，其中φ(β)是标准正态分布，$\mu$为随机序列$\{f_i\}$的期望值，$\sigma_f$为函数标准差。

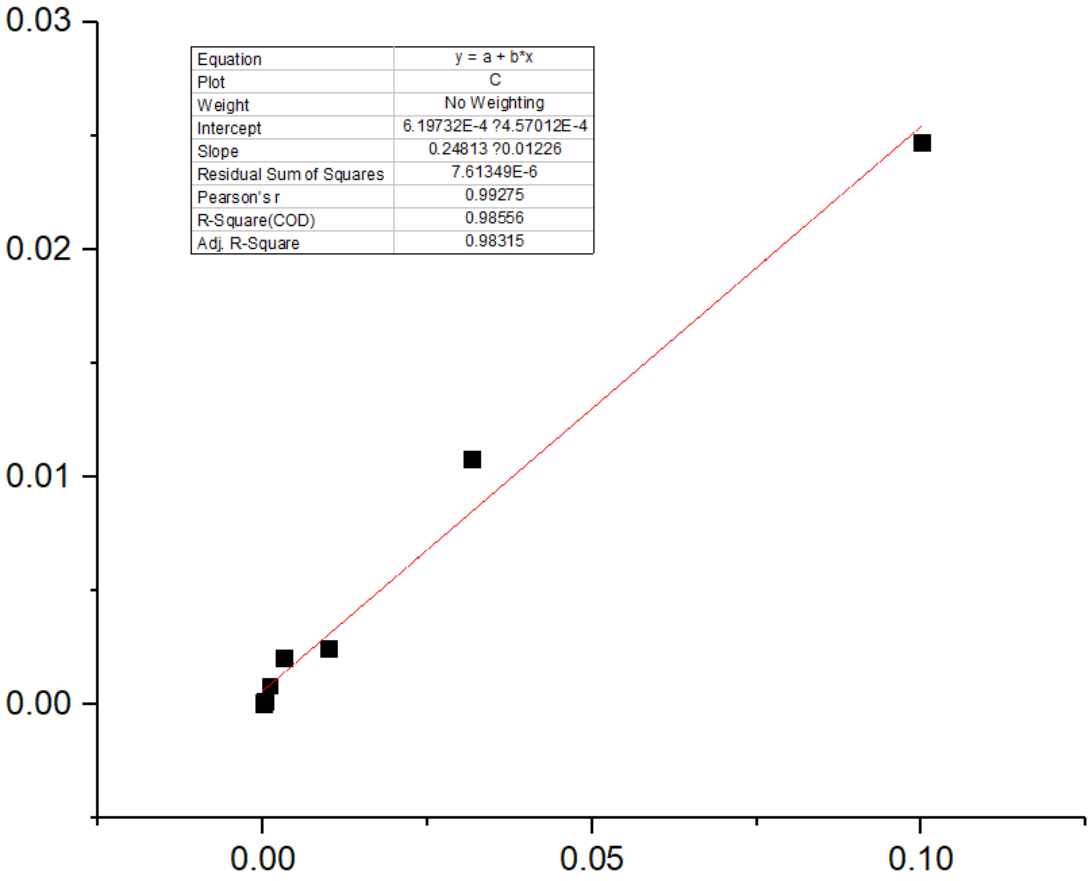
积分值$\mathrm{s} = \int_a^b f(x)dx = (b-a) < f >$，定义$< \mathrm{f} > = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$，积分值方差$\sigma_s^2 = var(s) = var\big((b-a) < f >\big) = (b-a)^2 var(< f >)$，积分值标准差$\sigma_s = (b-a)\sqrt{E(< f > -\mu)^2} = (\mathrm{b}-\mathrm{a})\frac{\sigma_f}{\sqrt{N}}$，上式说明，$\sigma_s$随着$\frac{1}{\sqrt{N}}$变化，那么以平方的方式增加样本点数，也只能线性缩小误差。

计算结果分析:

单变量积分

100	1.03500397	2.47416518e-002
1000	1.04579771	1.07937381e-002
10000	1.04826740	2.46968885e-003
100000	1.04620788	2.05952420e-003
1000000	1.04536779	8.40085559e-004
10000000	1.04520979	1.58003889e-004
100000000	1.04533831	1.28519060e-004
1000000000	1.04528712	5.11827056e-005

这个结果显然，因为取点数目的增加，误差也在减小，我们对于误差和取点进行线性拟合:



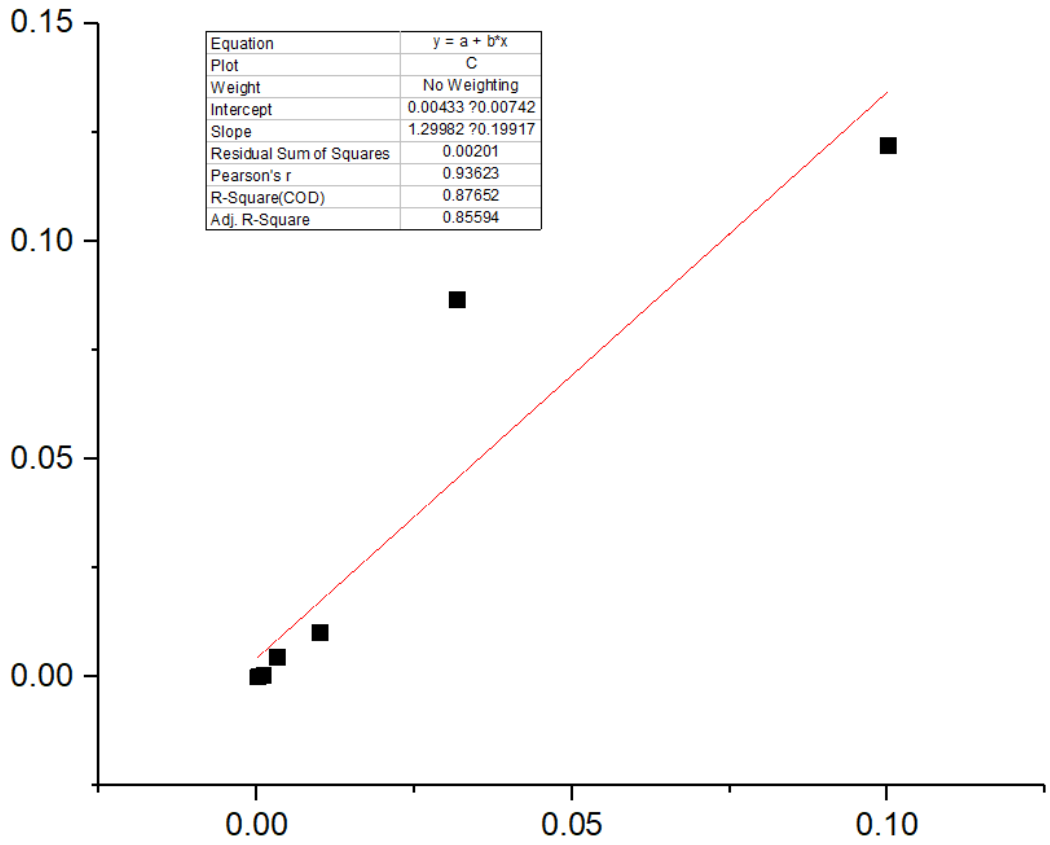
这是点的个数的根号分之一和误差之间的关系，可以看出线性关系还是比较明显的，而且 r 也很大。

多重积分

100,	2.639863685928643,	1.221392729903563e-001
1000,	2.553061707903078,	8.680197802556489e-002
10000,	2.563265615344028,	1.020390744094923e-002
100000,	2.558675404878800,	4.590210465227340e-003
1000000,	2.559188458489903,	5.130536111024320e-004
10000000,	2.559571619947336,	3.831614574334452e-004
100000000,	2.559519587078319,	5.203286901744875e-005
1000000000,	2.559465434322083,	5.415275623565918e-005

可以看出确实在取点个数增加以后，误差在减小，而且也较为严格的按照平方的规律在减小。

继续绘制类似的误差的图：



我们可以看到，线性的关系仍然较为明显。

实验总结：

1. 我们发现误差会随着计算点数目的增加而不断减小，但是速度会比较慢，这说明了 Monte Carlo 方法的缺点。
2. 在比较高的维度这个方法的误差变得更加规律了，而且收敛并不逊色一维的情况，就是误差关系的线性没有之前严格了。
3. 从这里可以发现 Monte Carlo 方法的几何意义，从一维上来看，当你取点变多的时候就用了更多的点来试探函数在各个定义域上的分布，这样子取点的数目的平均值以后，就更加接近几何平均值了，所以就可以用来模拟，而高维度也是如此。

程序：

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

main() {

    FILE *fp;

    fp=fopen("jieguo.txt","w");

    double a = 0, b = 1;//这里是设定两个数
```

```
    double jifen;//这里再定义两个实型的变量，其中的第一个是我们接下来要计算的积分值

    double jifen1[9];//用来计算误差

    srand(1);//要设置随机数，所以有这个随机数种子

    for (int j=1; j<10; j++) {

        int   A = pow(10, j);    // 随机点的个数

        double sum = 0;          // 这个用来对这么多的点的数值进行求和

        for (int i=0; i<A; i++) {//随机点的个数

            double x = 1.0*rand()/RAND_MAX;//任何一次对 rand 的调用，都将得到一个 0~RAND_MAX 之间的伪随机数这里，由于返回值是整型，所以还要加上一个 1.0

            sum += sqrt(x + sqrt(x));//在这里每次要计算被积函数的数值并且加很多次

        }

        jifen1[j]=(b-a)*sum/A;

        jifen = (b-a)*sum/A;//这里是 Monte Carlo 积分的最后的公式，而且显然取点个数增加精确度也会增加

        if(j>1)

        fprintf(fp,"%15d %25.8lf %25.8le\n", A, jifen, fabs(jifen-jifen1[j-1]));//最后输出取点的个数，积分值以及精确值和积分值之间的误差

    }

}
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

main() {
```

```
FILE *fp;
fp=fopen("jieguo1.txt","w");
int D = 5;       //  计算的积分的维数
double x_max[] = {7/10.0, 4/5.0, 9/10.0, 1.0, 11/10.0};   //积分的上确界
double x[D];   //设置一个有五个元素的数组
double jifen;
double jifen1[9];//用来计算误差
srand(100);//随机数的种子
for (int j=1; j<10; j++) {
    int   A = pow(10, j);   //  计算的次数
    double sum = 0;          //  用来计数函数值的求和
    for (int i=0; i<A; i++) {
        double fun_value = 6;     //  给函数值设置一个初值
        for (int l=0; l<D; l++) {//循环数组，取出积分范围内的随机点，然后求出函数值
            x[l] = x_max[l]*rand()/RAND_MAX;
            fun_value -= x[l]*x[l];
        }
        sum += fun_value;//每次循环都要把函数值给加上，这样就得到在这个 5 维空间内的 A 个随机点带入得到的函数值
    }
    for (int l=0; l<D; l++) {
        sum *= x_max[l];//然后按照多维的计算公式乘上积分上下限之差的乘积
```

```
            }

            jifen1[j]=sum/A;

            jifen = sum/A;//然后再按照公式就可以求出积分值了

            if(j>1)

            fprintf(fp,"%15d,%25.15lf,%25.15le\n", A, jifen,fabs(jifen-jifen1[j-1]));

        }

}
```

结果输出：

```
            100,    2.639863685928643,    1.221392729903563e-001
           1000,    2.553061707903078,    8.680197802556489e-002
          10000,    2.563265615344028,    1.020390744094923e-002
         100000,    2.558675404878800,    4.590210465227340e-003
        1000000,    2.559188458489903,    5.130536111024320e-004
       10000000,    2.559571619947336,    3.831614574334452e-004
      100000000,    2.559519587078319,    5.203286901744875e-005
     1000000000,    2.559465434322083,    5.415275623565918e-005
            100          1.03500397          2.47416518e-002
           1000          1.04579771          1.07937381e-002
          10000          1.04826740          2.46968885e-003
         100000          1.04620788          2.05952420e-003
        1000000          1.04536779          8.40085559e-004
       10000000          1.04520979          1.58003889e-004
      100000000          1.04533831          1.28519060e-004
     1000000000          1.04528712          5.11827056e-005
```