Project 1

苗睿中 11307110107

一．摘要：

本报告演示了计算机浮点数储存时的系统误差，使用计算机进行数值积分的算法误差，使用蒙特卡洛方法进行数值积分，确定性抽样和重要性抽样，以及计算高维积分时，蒙特卡洛方法和梯形法则的方法对比。

本报告中采用 Matlab 软件进行演示。

二．原理：

1. 计算机浮点数：

1985 年 IEEE 颁布了《Binary Floating Point Arithmetic Standard 754-1985》。2008 年，其更新版本《IEEE 754-2008》颁布，其中对计算机对实数的表示进行了规定，计算机中的实数可以用以下形式表示：

$$(-1)^s 2^{c-1023}(1+f)$$

其中，s，c，f分别用如下方式表示：

0 10000000011 1011100100010000000000000000000000000000000000000000000000

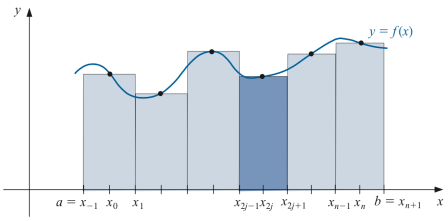
s c f

上述表达式中，$(-1)^s$代表实数的符号；2^{c-1023}代表指数部分，其范围在2^{-1023}到2^{1024}；$(1+f)$代表有效数字部分，因为 52 位二进制数对应 16 到 17 位十进制数，所以一般可以认为，这种表示下的实数有 16 为有效数字。

2. 数值积分方法：

（1）中点法(Midpoint rule)：

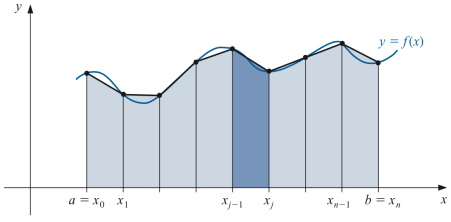
$$\int_a^b f(x)dx = 2h\sum_{j=0}^{n/2} f(x_{2j}) + \frac{b-a}{6}h^2 f''(\mu)$$



其中$h = x_{j+1} - x_j$，μ为(a,b)中的某点。二阶导数项代表误差。

（2）梯形法(Trapezoidal rule)：

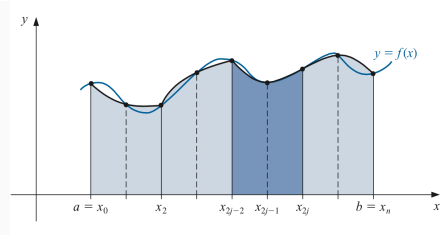
$$\int_a^b f(x)dx = \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right] - \frac{b-a}{12}h^2 f''(\mu)$$



可以看出，梯形法的误差项与中点法的误差项都以$h^2(\sim\frac{1}{n^2})$的速度收敛。

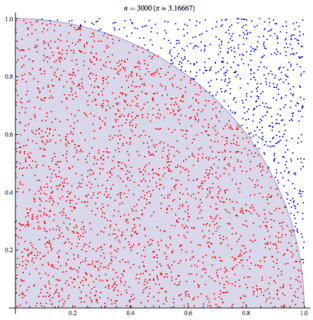
（3）辛普森法(Simpson's rule)：

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(b)\right] - \frac{b-a}{180}h^4 f^{(4)}(\mu)$$



辛普森法的误差项以$h^4(\sim\frac{1}{n^4})$的速度收敛。

3. 蒙特卡洛方法计算积分：



该方法在被积函数的定义域内进行随机抽样，并且在抽样点对被积函数值求平均，从而可得到积分的近似值。

这种方法有很多采样方式，对被积函数定义域进行均匀采样称为确定性采样，或非权重采样；若样本点满足某一特殊的分布，则称为重要性采样。

在确定性采样中，由中心极限定理可知，蒙特卡洛方法的收敛速度约为$(\sim\frac{1}{\sqrt{n}})$。可见，在一维情形下，该方法的收敛速度较慢，但可证明对于高维积分，该方法优于前几种方法。

三．程序及结果：

1. 计算机浮点数误差：

从上述原理可以看出，计算机的指数部分范围在2^{-1023}到2^{1024}之间，有效数字为 16 个，因此当我们储存一个较大的数字，例如$2^{1000}(\sim 10^{31})$时，计算机只保存其前面 16 个有效数字，后面的有效数字将被舍去。例如，我们可以输入 matlab 命令：

```
>> a=2^1000+1;
>> b=2^1000;
>> a-b
ans=
        0
```

可见，虽然数学上 a 比 b 大 1，但由于计算机有效数字位数有限，因此个位数上的 1 被舍去，a-b 的结果为 0。

2. 数值积分方法：

我们先建立三个 Matlab function 来进行三种数值方法的运算：

（1）中点法：

```
function integral= midpoint( f,a,b,n )
  integral=0;
  h=(b-a)/n;
  for i=1:2:(n-1)
     integral=integral+2*h*f(a+h*i);
  end
end
```

（2）梯形法：

```
function integral = trapezoidal( f,a,b,n )
  h=(b-a)/n;
  integral=0;
  for i=1:n
     integral=integral+h*(f(a+(i-1)*h)+f(a+i*h))/2;
  end
end
```

（3）辛普森法：

```
function integral= simpson( f,a,b,n )
  h=(b-a)/n;
  sum=f(a)+f(b);
  for i=2:2:n-2
     sum=sum+2*f(a+i*h);
  end
  for i=1:2:n-1
     sum=sum+4*f(a+i*h);
  end
  integral=h/3*sum;
end
```

上述三个函数中，n 代表小区间的个数，h 代表每个小区间的跨度，a 和 b 分别是积分的上下限，f 是被积函数。

接下来，我们进行$f(x) = x^5 + x^3 + x$在区间(0,6)的积分。该积分有解析解 8118。为了比较这三种计算方法。我们定义 demo 函数：

```
function [ ] = demo( method,f,a,b )
 truevalue=integral(f,a,b);       %integral is a built-in function of matlab.
 %This gives the true value of the integral being estimated.
 fprintf('    n          h          results          error\n');
 fprintf('-------------------------------------------------\n');
 for i=1:7
    n=10^i;
    value=method(f,a,b,n);        %We select a method
    fprintf('%7d  %12.7f  %12.8f  %12.8f\n',n,(b-a)/n,value,abs(value-truevalue));
```

```
  end
end
```

输入 Matlab 命令：

```
>> f=@(x) x.^5+x.^3+x;
>> method=@midpoint;       %Midpoint rule is selected.
>> demo(method,f,0,6)
```

则程序输出：

```
      n             h             results              error
 ------------------------------------------------------------------------------
         10    0.6000000   7728.16320000  389.83680000
        100    0.0600000   8114.04774432    3.95225568
       1000    0.0060000   8117.96047205    0.03952795
      10000    0.0006000   8117.99960472    0.00039528
     100000    0.0000600   8117.99999605    0.00000395
    1000000    0.0000060   8117.99999996    0.00000004
   10000000    0.0000006   8118.00000000    0.00000000
```

接下来，我们来看梯形法的结果，输入：

```
>> f=@(x) x.^5+x.^3+x;
>> method=@trapezoidal;       %Trapezoidal rule is selected.
>> demo(method,f,0,6)
```

程序输出：

```
      n             h             results              error
 ------------------------------------------------------------------------------
         10    0.6000000   8315.25120000  197.25120000
        100    0.0600000   8119.97636112    1.97636112
       1000    0.0060000   8118.01976400    0.01976400
      10000    0.0006000   8118.00019764    0.00019764
     100000    0.0000600   8118.00000198    0.00000198
    1000000    0.0000060   8118.00000002    0.00000002
   10000000    0.0000006   8118.00000000    0.00000000
```

辛普森法的结果，输入：

```
>> f=@(x) x.^5+x.^3+x;
>> method=@simpson;       %Simpson rule is selected.
>> demo(method,f,0,6)
```

程序输出：

n	h	results	error
10	0.6000000	8119.55520000	1.55520000
100	0.0600000	8118.00015552	0.00015552
1000	0.0060000	8118.00000002	0.00000002
10000	0.0006000	8118.00000000	0.00000000
100000	0.0000600	8118.00000000	0.00000000
1000000	0.0000060	8118.00000000	0.00000000
10000000	0.0000006	8118.00000000	0.00000000

从上述结果可以看出，中点法和梯形法误差的收敛速度处于一个量级，梯形法的误差基本等于中点法的一半；辛普森法的误差收敛速度明显快于前两种方法。

3. 计算稳定性：

一般地，随着计算机运算次数的增多，截断误差（round-off error）会随之增长。通过查阅文献，可以了解到，上述计算积分的方法有一个优良的性质：当小区间划分的很细时，计算的次数会增加，但截断误差不会因为区间的增多而增大。其证明如下：

设我们用$\tilde{f}(x_i)$来近似$f(x_i)$，所带来的截断误差用e_i来表示，则：

$$f(x_i) = \tilde{f}(x_i) + e_i$$

若我们用梯形法（另两种方法的证明类似），则总误差可以表示为：

$$e(h) = \frac{h}{2}\left[e_0 + 2\sum_{i=1}^{n-1} e_i + e_n\right]$$

$$e(h) \le \frac{h}{2}\left[|e_0| + 2\sum_{i=1}^{n-1} |e_i| + |e_n|\right]$$

设所有的误差的上限为ε（对于单次计算，截断误差的上限），则有：

$$e(h) \le \frac{h}{2}\left[\varepsilon + 2\sum_{i=1}^{n-1} \varepsilon + \varepsilon\right] = \frac{h}{2} \times 2n\varepsilon = nh\varepsilon = (b-a)\varepsilon$$

上式用到了等式$nh = (b-a)$，可见$e(h)$并不依赖于n。这给我们的启示是：在使用上述方法计算积分时，在计算机可以表示的实数范围内，可以将n设的很大而无需考虑因此带来的截断误差。

4. 蒙特卡洛方法计算积分：

（1）确定性抽样：

类似于前三个方法，我们定义一个 Matlab function 来进行蒙特卡洛方法的计算：

```
function I = MonteCarlo( f,a,b,n )
  I=0;
  for i=1:n
     p=rand()*(b-a)+a;
     I=I+f(p);
  end
  I=I*(b-a)/n;
end
```

在 Matlab 命令行输入：

```
>> f=@(x) x.^5+x.^3+x;
>> method=@MonteCarlo;        %Monte Carlo method is selected.
>> demo(method,f,0,6)
```

程序输出：

n	h	results	error
10	0.6000000	4161.20651077	3956.79348923
100	0.0600000	9082.31817305	964.31817305
1000	0.0060000	7818.07499387	299.92500613
10000	0.0006000	8319.03156302	201.03156302
100000	0.0000600	8137.96689934	19.96689934
1000000	0.0000060	8106.04287463	11.95712537
10000000	0.0000006	8126.54985382	8.54985382

可以看出，蒙特卡洛方法计算一维积分的收敛速度慢于前三种方法。

（2）重要性抽样：

从被积函数$f(x)=x^5+x^3+x$的形式可以看出，随着x增大，$f(x)$也随之增大，即被积函数的在x较大时，权重较大。若将抽样密度函数改为：

$$p(x)=\frac{2}{b^2\text{-}a^2}x,\quad x\in(a,b)$$

则

$$\int_a^b f(x)dx=\int_a^b \frac{f(x)}{p(x)}p(x)dx=\int_a^b \frac{f(x)}{p(x)}dP(x)$$

即计算$\frac{f(x)}{p(x)}$在$x\sim p(x)$分布时的平均值。

设r为$(0,1)$上满足均匀分布的随机变量，则

$$x = \sqrt{r(b^2\text{-}a^2) + a^2}$$

即是满足$x\sim p(x)$的随机变量。定义 Matlab function：

```
function I = MonteCarloI( f,a,b,n )
 I=0;
  for i=1:n
      p=sqrt(rand()*(b^2-a^2)*a^2);       % p ~ p(x) distribution
      if p~=0
         I=I+f(p)*(b^2-a^2)/p/2;        %  f(x)/p(x)
      else
         i=i-1;
      end
  end
 I=I*(b-a)/n;
end
```

输入指令：

```
>> f=@(x) x.^5+x.^3+x;
>> method=@MonteCarloI;        %Monte Carlo method is selected.
                               %Improtance sampling
>> demo(method,f,0,6)
```

程序输出：

```
      n             h             results              error
 ------------------------------------------------------------------------
        10     0.6000000   6650.78161749  1467.21838251
       100     0.0600000   8875.05078529   757.05078529
      1000     0.0060000   8171.13807805    53.13807805
     10000     0.0006000   8078.37037751    39.62962249
    100000     0.0000600   8110.99528619     7.00471381
   1000000     0.0000060   8112.80211787     5.19788213
  10000000     0.0000006   8120.94646061     2.94646061
```

可以看出，采用重要性采样后，该方法的收敛速度有一定的提高。

（3）高维积分中的蒙特卡洛方法：

在上面的讨论中，我们说过在计算高维积分时，蒙特卡洛方法优于前三种方法，这里我们通过计算函数

$$\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1 {x_1}^2+{x_2}^2+{x_3}^2+{x_4}^2+{x_5}^2 dx_1dx_2dx_3dx_4dx_5$$

该积分有解析解5/3，我们使用梯形法和蒙特卡罗法分别计算该积分的值。

a. 梯形法：
在 Matlab 命令行输入：

```
a=[0,0,0,0,0];
b=[1,1,1,1,1];
g=@(x1,x2,x3,x4,x5) x1.^2+x2.^2+x3.^2+x4.^2+x5.^2;        % Define the function
fprintf('    n          results          error\n');
fprintf('-------------------------------------------\n');
for s=5:15
I=0;
h=(b-a)/s;
delta=cumprod(h);
delta=delta(length(a));
for i=1:s
for j=1:s
for k=1:s
for l=1:s
for m=1:s
  I=I+f(a(1)+(i-1)*h(1),a(2)+(j-1)*h(2),a(3)+(k-1)*h(3),a(4)+(l-1)*h(4),a(5)+(m-
1)*h(5))*delta/2;
  I=I+f(a(1)+i*h(1),a(2)+j*h(2),a(3)+k*h(3),a(4)+l*h(4),a(5)+m*h(5))*delta/2;
end
end
end
end
end
fprintf('%7d   %12.8f  %12.8f\n',s^5,I,abs(I-5/3));
end
```

程序输出：

```
    n            results          error
-------------------------------------------------
   3125      1.70000000    0.03333333
   7776      1.68981481    0.02314815
  16807      1.68367347    0.01700680
  32768      1.67968750    0.01302083
  59049      1.67695473    0.01028807
 100000      1.67500000    0.00833333
 161051      1.67355372    0.00688705
```

```
 248832    1.67245370    0.00578704
 371293    1.67159763    0.00493097
 537824    1.67091837    0.00425170
 759375    1.67037037    0.00370370
```

b. 蒙特卡洛方法：
在 Matlab 命令行输入：

```
fprintf('    n          results         error\n');
fprintf('-------------------------------------------\n');
for s=5:15
   n=s^5;
I=0;
delta=cumprod(b-a);
delta=delta(length(a));
for i=1:n
   p=rand(1,5).*(b-a)+a;
   I=I+f(p(1),p(2),p(3),p(4),p(5));
end
I=I/n*delta;
fprintf('%7d   %12.8f  %12.8f\n',n,I,abs(I-5/3));
end
```

程序输出：

```
     n             results           error
-------------------------------------------------
    3125      1.67395103    0.00728436
    7776      1.65586347    0.01080320
   16807      1.66929968    0.00263301
   32768      1.67164742    0.00498076
   59049      1.66483891    0.00182775
  100000      1.66593597    0.00073070
  161051      1.66763108    0.00096442
  248832      1.66579928    0.00086739
  371293      1.66716123    0.00049456
  537824      1.66547183    0.00119483
  759375      1.66714774    0.00048108
```

可以看出，在对于这个五维度的积分，蒙特卡洛方法的收敛速度快于梯形法。当积分的维度变得更高时，蒙特卡洛方法的优越性会更明显。

需要注意的是，蒙特卡洛方法的正确性是建立在大数定律和中心极限定律上的，因此蒙特卡洛方法的收敛性是依概率收敛，在实际应用中应该注意这一点。

四．参考文献：

1. Richard L. Burden and J. Douglas Faires, *Numerical Analysis*, Ninth Edition.
2. Christian P. Robert and George Casella, *Monte Carlo Statistical Method*, Second Edition.