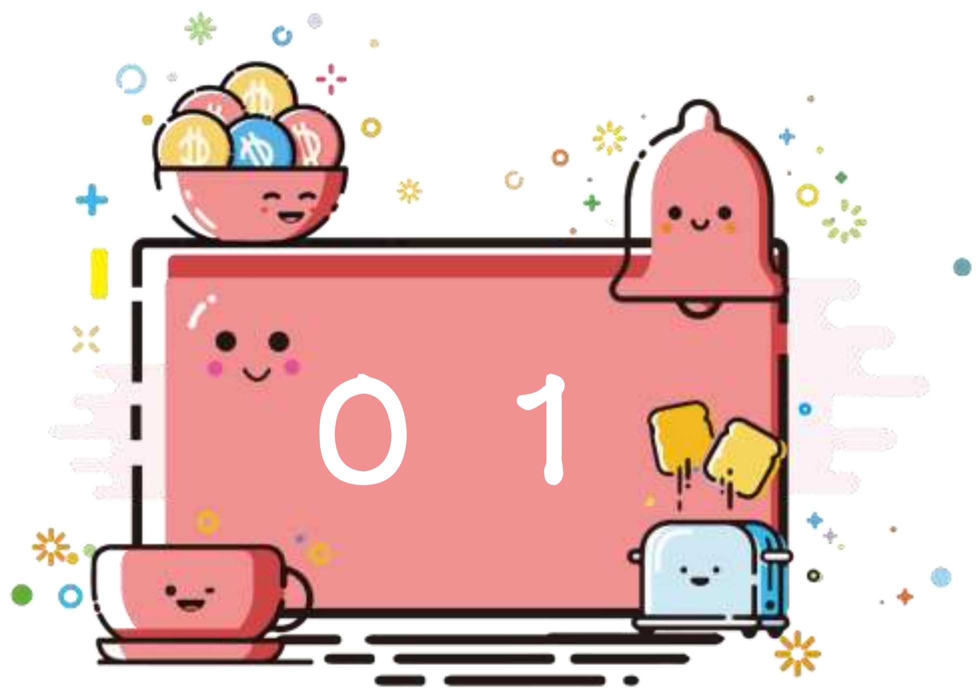
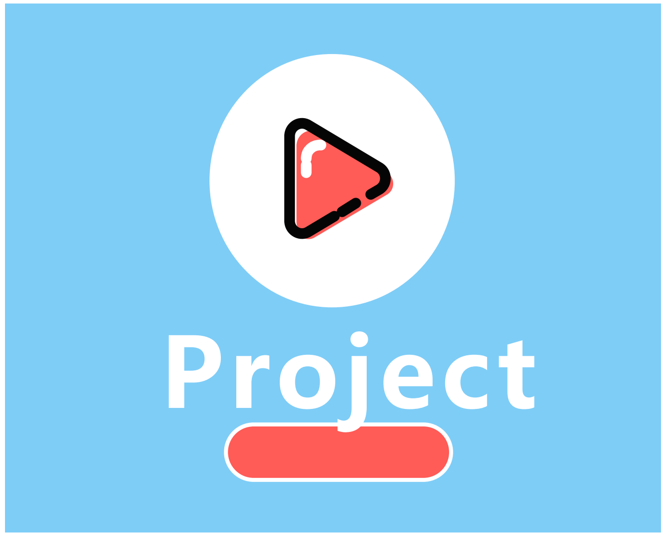
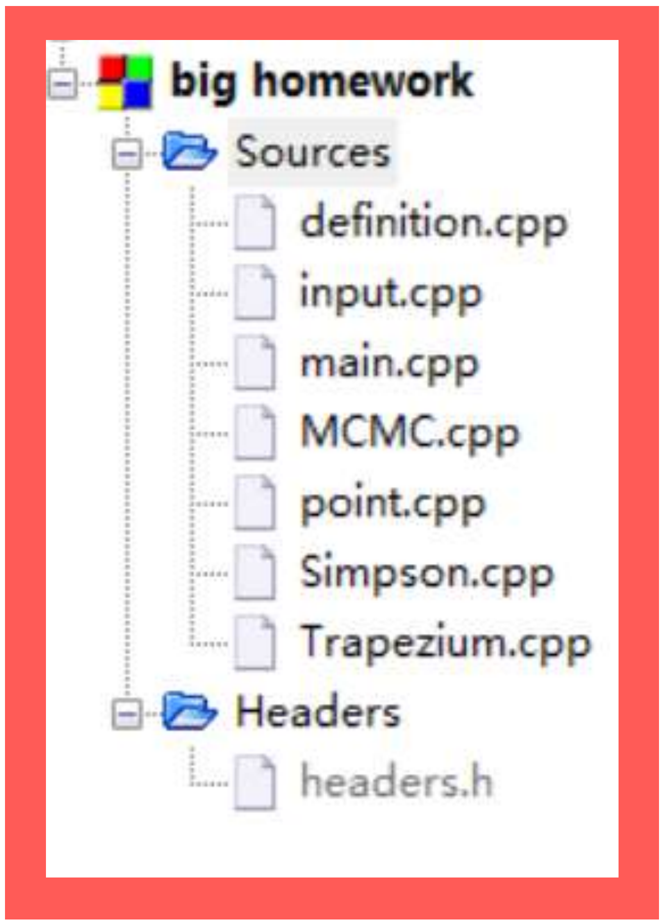
C++定积分计算包


目录



整体介绍





①Main.cpp：主函数板块。

②Input.cpp：求解f(X)。

③Point.cpp：判断瑕点和瑕积分的敛散性。

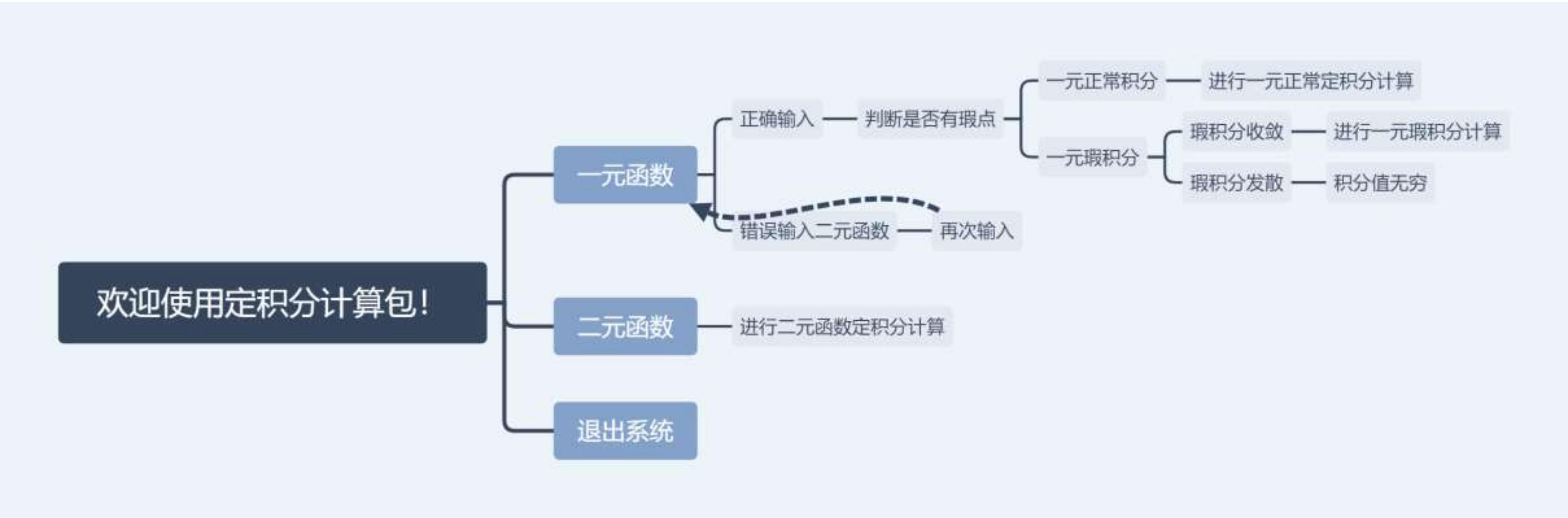
④Definition.cpp：定义法求解定积分。

⑤Trapezium.cpp：梯形法求解定积分。

⑥Simpson.cpp：辛普森算法求解定积分。

⑦MCMC.cpp：蒙特卡洛投点法和蒙特卡洛平均值法求解定积分。

⑧Headers.h：头文件。


欢迎使用定积分计算包!
一元函数
正确输入
判断是否有瑕点
一元正常积分
进行一元正常定积分计算
一元瑕积分
瑕积分收敛
进行一元瑕积分计算
瑕积分发散
积分值无穷
错误输入二元函数
再次输入
二元函数
进行二元函数定积分计算
退出系统

具体功能

目录

1

Input函数

——求解f(x)

Solve函数

中心思想

用string型变量orifunc储存用户输入的函数表达式

根据运算符的优先级，识别不同的运算符operat[i]，并对operat[i]两边的number[i]和number[i+1]进行运算。清除参与运算的operat[i]并将后面的元素往前推一位；将number[i]替换为运算结果，清除参与运算的number[i+1]，并将number[i+1]后面的元素往前推一位

string： x^2+2*x/5

X=3 ↓

string： 3^2+ 2*3/5

↓ judge函数判断数字/运算符

number(vector):3 2 2 3 5

operat(vector): ^ + * /

↓

number(vector):9 2 3 5

operat(vector): + * /

↓

number(vector):10.2

operat(vector):

Solve函数

```
sprintf( transferx , "%f" , x );
sprintf( transfere , "%f" , 2.71828 );
sprintf( transferp , "%f" , 3.14159);


for( i = 0 ; orifunc[i] != '\0' ; i++){
        if( orifunc[i] == 'x' ){
            orifunc.insert( i + 1 , transferx );
            orifunc.erase( i , 1 );
        }
        else if( orifunc[i] == 'e' ){
            orifunc.insert( i+1 , transfere );
            orifunc.erase( i , 1 );
        }
        else if( orifunc[i] == 'p' ){
            orifunc.insert( i+1 , transferp );
            orifunc.erase( i , 1 );
        }
    }
}
```

string： x^2+2*x/5

X=3

string： 3^2+ 2*3/5

judge函数判断数字/运算符

number(vector):3 2 2 3 5

operat(vector): ^ + * /

number(vector):9 2 3 5

operat(vector): + * /

number(vector):10.2

operat(vector):

Solve函数

```
 while( judge (orifunc[i]) ) {
if( orifunc[i] >= '0' && orifunc[i] <= '9' ) {
 //计算整数部分
        sum = sum * 10 + ( orifunc[i] - 48 ) ;
        i++ ;
     }
     else if( orifunc[i] == '.’  )
//计算小数部分,并暂存在decimal/j中
     {
        int j = 1 ;
        decimal = 0 ;
        i++ ;
        while( judge ( orifunc[i] ) )
        {
           decimal = decimal * 10 +
( orifunc[i] - 48 ) ;
           j = j * 10.0 ;
           i++ ;
        }
        sum = sum + decimal / j ;
     }
   }
```

string： x^2+2*x/5

X=3

string： 3^2+ 2*3/5

judge函数判断数字/运算符

number(vector):3 2 2 3 5

operat(vector): ^ + * /

number(vector):9 2 3 5

operat(vector): + * /

number(vector):10.2

operat(vector):

Solve函数

```
if( operat[i] == '^' ) {
        temp = pow ( number[i] , number[i+1] ) ;
        number[i] = temp ;
        for( j = i+1 ; j < number.size() ; j++ )
            number[j] = number[j+1] ;
        for( j = i ; j < operat.size() ; j++ )
            operat[j] = operat[j+1] ;
            }
```

string： x^2+2*x/5

X=3

string： 3^2+ 2*3/5

judge函数判断数字/运算符

number(vector):3 2 2 3 5

operat(vector): ^ + * /

number(vector):9 2 3 5

operat(vector): + * /

number(vector):10.2

operat(vector):

如何保证数字与运算符交替储存在string型数组中并依次进行计算?

难点一：负数的运算

String： x^2+2*x/5

X=-3

string： -3^2+ 2*-3/5

number(vector):0 3 2 2 -3 5

operat(vector): - ^ + * /

Solve函数

```
flag=0
if( !judge(orifunc[i]) ){
        if( judge( orifunc[i-1] ) ){
//若两个运算符相邻
        operat.push_back( orifunc[i] ) ;
//则将第一个运算符储存入operat数组
        i ++ ;
        }
    ……
while( judge (orifunc[i]) ) { //计算数字
    ……
    if( judge( orifunc[a] ) ){
    if( flag ) sum = - sum; //取相反数
    number.push_back( sum ) ;}}
```

难点二：相邻符号优先级相同

Solve函数

String： x*1*2+3

X=0

string： 0*1*2+3

number(vector):0 1 2 3

operat(vector): * * +

```
for( i = 0 ; i < operat.size() ; i++ ) {
        if( operat[i] == ‘*' ) {
            temp = number[i] * number[i+1] ;
            number[i] = temp ;
            for( j = i + 1 ; j < number.size(); j++ )
                number[j] = number[j+1];
            for( j = i ; j < operat.size() ; j++ )
                operat[j] = operat[j+1] ;
    }
```

难点二：相邻符号优先级相同

Solve函数

string： 0*1*2+3

i=0：

number(vector):0 1 2 3

operat(vector): * * +

i=1：

number(vector):0 2 3

operat(vector): * +

结束循环：

number(vector):0 5

operat(vector): *

```
for( i = 0 ; i < operat.size() ; i++ ) {
        if( operat[i] ==  ‘*' ) {
            temp = number[i] * number[i+1] ;
            number[i] = temp ;
            for( j = i + 1 ; j < number.size(); j++ )
                number[j] = number[j+1];
            for( j = i ; j < operat.size() ; j++ )
                operat[j] = operat[j+1] ;
    }
```

难点二：相邻符号优先级相同

Solve函数

string: 0*1*2+3

i=0：

number(vector):0 1 2 3

operat(vector): * * +

i=1：

number(vector):0 2 3

operat(vector): * +

结束循环:

number(vector):0 5

operat(vector): *

```
for( k = 0 ; k < operat.size() ; k++ ) {
    for( i = 0 ; i < operat.size() ; i++ ) {
            if( operat[i] ==  '*' ) {
                temp = number[i] * number[i+1] ;
                number[i] = temp ;
                for( j = i + 1 ; j < number.size(); j++ )
                    number[j] = number[j+1];
                for( j = i ; j < operat.size() ; j++ )
                    operat[j] = operat[j+1] ;
    }
}
```

难点二：相邻符号优先级相同

string： 0*1*2+3

i=0：

number(vector):0 1 2 3

operat(vector): * * +

i=1：

number(vector):0 2 3

operat(vector): * +

结束循环：

number(vector):0 5

operat(vector): *

Solve函数

number(vector):0 1 2 3

operat(vector): * * +

number(vector):0 2 3

operat(vector): * +

number(vector):0 3

operat(vector): +

number(vector):3

operat(vector):

mainsolve函数

难点三：括号的运算

Transfer、logtransfer、tritransfer函数

String：(x-1)^2+sin(x)-log(e,x)

1、括号定位

2、储存括号内的内容（string型），根据括号运算的类型，对其调用solve函数，得到double型结果

3、将double型结果转换为char型，并替换掉方框内的字符，插入orifunc中

String：0^2+0.84147-0

```
while( orifunc[j] != ',' && j < righthalf ){
        lhfunction.push_back( orifunc[j] ) ;
        j++;
    }//碰不到，和）时
    lmid = solve( lhfunction , x ) ;
    if( orifunc[j] == ',' ){
        j++;
        while( j < righthalf ){
            rhfunction.push_back( orifunc[j] ) ;
            j++ ;
        }//如果碰到",",在碰到）之前都存入rhfunction中
        rmid = solve( rhfunction , x ) ;
        orifunc = logtransfer( lmid , rmid , orifunc ,
lefthalf , righthalf ) ;
    }//碰到","时
    else
        orifunc = tritransfer( lmid , orifunc , lefthalf ,
righthalf ) ;
    lhfunction.clear() ;
    rhfunction.clear() ;
    return orifunc ;
```

难点三：括号的运算

Transfer、logtransfer、tritransfer函数

String：(x-1)^2+sin(x)-log(e,x)

1、括号定位

2、储存括号内的内容（string型），根据括号运算的类型，对其调用solve函数，得到double型结果

3、将double型结果转换为char型，并替换掉方框内的字符，插入orifunc中

String：0^2+0.84147-0

String： log(e , x)

String: lhfunction rhfunction

solve()函数

double: lmid rmid

double: logvalue = log(rmid)/log(lmid)

ininf()&sprint()

string: value

①inf:定义为1000000000

②-inf:定义为-1000000000

③else:利用sprintf()进行转换

插入orifunc(string型)

难点三：括号的运算

Transfer、logtransfer、tritransfer函数

String：(x-1)^2+sin(x)-log(e,x)

1、括号定位
2、储存括号内的内容（string型），根据括号运算的类型，对其调用solve函数，得到double型结果
3、将double型结果转换为char型，并替换掉方框内的字符，插入orifunc中

String：0^2+0.84147-0

String： tan(x)

String: lhfunction

double: mid

判别()前几位是否为（反）三角函数

是

否

以asin()为例：

```
for( h = lefthalf - 4 ; h < lefthalf ; h++ )
        check.push_back( orifunc[h] ) ;
if( check == “asin” )    trivalue = asin( mid ) ;
```

double: trivalue

string: value

插入orifunc(string型）

lastsolve 函 数

```
for(k = 0; k < orifunc.size(); k++) {
        for(i = 0; i < orifunc.size(); i++) {
            if(orifunc[i] == '|') {
                leftabs = i ;
                j = i + 1 ;
                while( orifunc[j] != '|' ) {

absfunction.push_back( orifunc[j] ) ;
                    j++ ;
                    rightabs = j ;
                }
                absmid =
mainsolve( absfunction , x ) ;
                orifunc = abstransfer( absmid ,
orifunc , leftabs , rightabs ) ;
                absfunction.clear() ;
```

binarysolve函数

```
sprintf( transfery , "%f" , y ) ;//将double型数字转换为char型数字，储存在transferx中
    for(i = 0 ; orifunc[i] != '\0'; i++){
        if(orifunc[i] == 'y'){
            orifunc.insert( i + 1 , transfery ) ;
            orifunc.erase( i , 1 ) ;
        }//将orifunc中的y替换为储存在transfery中的char型数字
    }
    answer = lastsolve( orifunc , x ) ;
```

2

point函数

——瑕积分

01 判断瑕点

判 断 瑕 点

```
class Point{
public:
    int point( string orifunc , double down , double up);
    double impro_Definition( string orifunc , double down , double up );
    double impro_Simpson( string orifunc , double down , double up );
    double impro_Trapezoid( string orifunc , double down , double up );
    double impro_Montecarlo( string orifunc , double down , double up );
    vector<double> storepoint;
};
```

01 判断瑕点

判 断 瑕 点

```
for( i = 0 ; i <= copies ; i ++ ){
    if ( lastsolve(orifunc , number) > DBL_MAX
      || lastsolve(orifunc , number) < -DBL_MAX ){
      storepoint.push_back(number);
      number = number + 1 ;
    }
    else
      number = number + minizone;
  }
```

01 判断瑕点

判 断 瑕 点

```
for( i = 0 ; i <= copies ; i ++ ){
    idx=orifunc.find("tan");//在orifunc中查找tan函数
    if(idx == string::npos )//不存在。
      pin = 3000000 ;
     else//存在。
      pin = 1000 ;
    if ( lastsolve( orifunc , number ) > pin
           || lastsolve( orifunc , number ) < -pin ){
       storepoint.push_back( number ) ;
       number = number + 0.1 ;
    }
    else
       number = number + minizone ;
 }
```

02

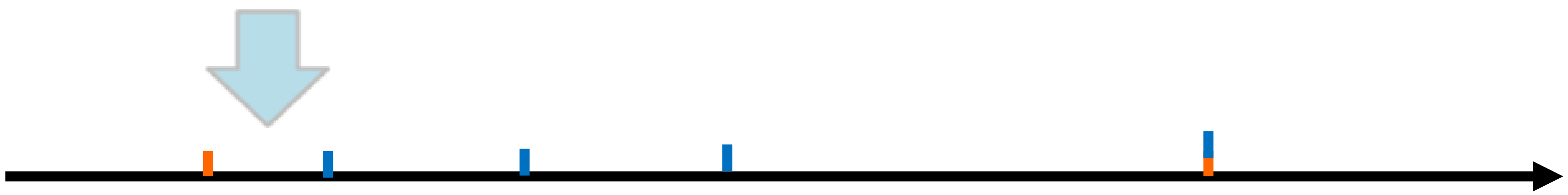
瑕积分与其收敛性的判断

思路：利用瑕点将积分区域划分为许多小区域，再判断每一个小区域的瑕积分是否为收敛的，如果有一个小区域的瑕积分发散，那么就认定整个积分区间的瑕积分是发散的。每一个小区域可以分为三个类型，一个类型是上限为瑕点，下限不为瑕点，一个类型是下限为瑕点，上限不为瑕点，最后一个类型是上下皆为瑕点的情况。

判断是否收敛：如果瑕积分收敛，那么在越来越靠近瑕点时，积分的变化程度不会过大，但如果瑕积分发散，在越来越靠近瑕点时，积分的变化程度会过大。在此处我们采用定义法来计算积分值，由定义法的计算可知，由于我们分割区间的长度不可能无穷大，故而在存在瑕点的那个区间所矩形近似的面积一定会是无穷，无论此积分是否发散，故而我们不采取判断积分值是否为无穷的方法来判断瑕积分是否收敛的问题，我们只靠判断它的在接近瑕点时积分的变化程度来判断瑕积分是否收敛。

02

瑕积分与其收敛性的判断



02

瑕积分与其收敛性的判断

while(time < 9){//使积分区间逼近瑕点八次
anothermid = Definition(orifunc , storepoint[i] + ase , storepoint[i+1] - ase) ;
if(!isinf(anothermid)) //由于随着区间的逼近，临界的值可能再一次超过计算机上限，从使得计算机将值记为inf，我们将这种情况的求得的积分值删去；
test.push_back(anothermid) ;
ase = ase * 0.1 ;//求得越来越逼近瑕点的积分值
time ++ ;
}
ase = 0.001 ;
time = 1 ;//变为原来的ase和time以便下一次循环的使用
for(j = 0 ; j < test.size() - 1 ; j ++){
if(abs(test[j + 1] - test[j]) > abs(test[j]))//用 test[j + 1] - test[j]) > abs(test[j])来判断随着逼近瑕点时积分的值是否变化很大，如果变化较小则认定为收敛，若变化大于自身则认为该瑕积分是发散的，因为在瑕积分发散时，积分区域的一小点变化就会引起积分较大的变化。

遇到的问题

瑕积分如果用定义法来算，由于划分的区域间的距离不够小，在瑕点的积分值都会趋于无穷，可瑕积分却有收敛和发散的情况，收敛时可以计算出值，故而判断瑕积分是否收敛是一个较难问题。

解决的方法

通过不断靠近瑕点的就得到许多积分值，并储存至test中，test比较每两个积分值之间变化是否很大来判断是否收敛

03

瑕积分的计算

在主函数中我们已经利用Point类中的point函数判断出了函数是否收敛，并且类中的storepoint数组的值改变，即储存了函数中所有的瑕点。故而下面的函数只用于计算函数收敛情况下的函数值。

主要思想:

将积分区间按瑕点划分为许多个小区间，由于是收敛的积分，故而小区间瑕积分和其很接近的一个区间的积分的值应该相差不大，故而可以用此区间的值代替瑕积分，这样我们就避开了两端的瑕点，使得积分可求，不会因瑕点的函数值为无穷而导致计算机不能够运算。

3 定义法

中心思想

定积分中“分割，近似求和，取极限”的思想，利用黎曼和求积分

```
double Definition( string orifunc , double down , double up ){
    ofstream fout;
    string path = "C:\\Users\\HP\\Desktop\\output.txt";
    fout.open(path,ios::app);
    int number , i ;
    double answer , delta = 0.0001 ;
    answer = 0;//用于保存累加得到的值
    number = (up - down)/delta;//计算总共需要分成多少份
    for( i = 0 ; i <number ; i ++ )
        answer = answer + delta * lastsolve( orifunc , down + delta * i ) ;
    fout.close();
    return answer ;
}
```

double Definition::dow

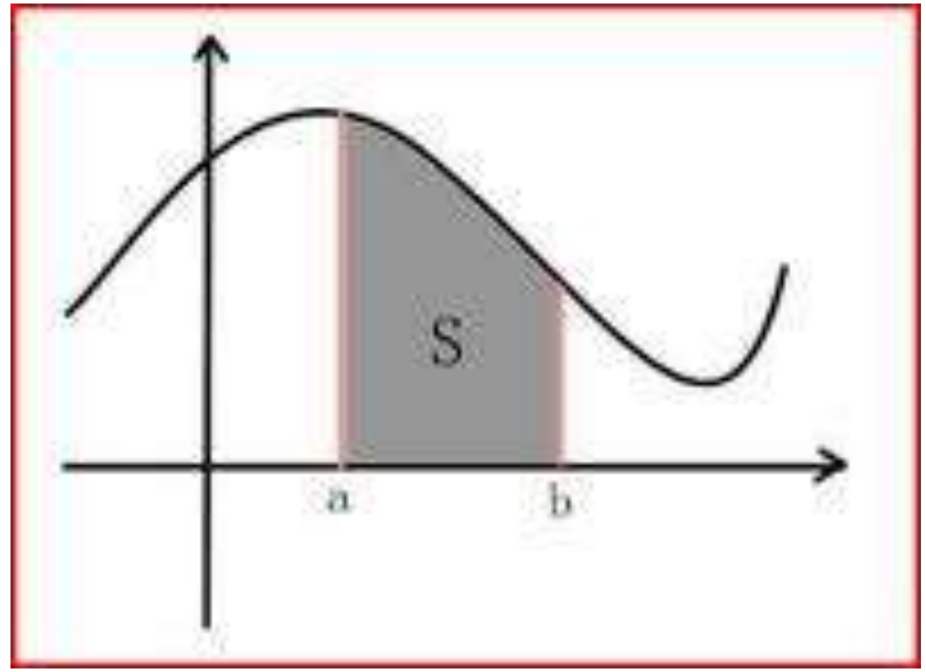
4

梯形法

数学法基础

代码讲解

数 学 基 础



假设被积函数为f(x)，积分区间为[a,b]，把区间[a,b]等分成n个小区间，各个区间的长度为step，即step=(b-a)/n，称之为“步长”。根据定积分的定义及几何意义，定积分就是求函数f(x)在区间[a,b]中图线下包围的面积。将积分区间n等分，各子区间的面积近似等于梯形的面积，面积的计算运用梯形公式求解，再累加各区间的面积，所得的和近似等于被积函数的积分值n越大，所得结果越精确。以上就是利用复合梯形公式实现定积分的计算的算法思想。

复合梯形公式：

$$T_n = \frac{step}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right]$$

代码解析

梯形法

求解一元定定积分

```
double Trapezoid( string orifunc , double down , double
up ){
    int number , i ;
double answer , delta  ;
 delta = 0.001。
answer = 0;
    number = (up - down)/delta;//确定循环次数
    for( i = 0 ; i <= number ; i ++ ){
        answer = answer + ( lastsolve( orifunc , down + delta *
i ) + lastsolve( orifunc , down + delta * (i + 1) ) * delta / 2 ) ;
    }
```

利用梯形面积计算公式，计算每个小梯形的面积。

令down + delta * i 点对应的函数值lastsolve(orifunc , down + delta * i)为梯形的上底；令down + delta * (i + 1)点对应的函数值lastsolve(orifunc , down + delta * (i + 1))为梯形的下底。

梯形的高为 delta，则小梯形的面积为lastsolve(orifunc , down + delta * i) + lastsolve(orifunc , down + delta * (i + 1)) * delta / 2。

利用for循环计算每个小梯形的面积并相加的得总面积为answer；所以用梯形法求得的一元定积分值为 answer。

代码解析

平均值法

求解二元定积分

```
double Trapezoid_branry(string orifunc,double downx,double upx,double downy,double upy)
{
    int numberx , numbery , i , j ;
    double answer , h1 , h2 , delta = 0.001 ;
    numberx = (upx - downx)/delta ;
    numbery = (upy - downy)/delta ;
    for( i = 0 ; i <= numberx ; i ++ ){
        for( j = 0 ; j <= numbery ; j ++ ){
            h1 = binarysolve( orifunc , downx + delta * i , downy + delta * j ) ;
            h2 = binarysolve( orifunc , downx + delta * i , downy + delta * (j + 1 )) ;
            answer = answer +( (h1 + h2)  * delta / 2)*delta ;
        }
    }
return  answer ;
}
```

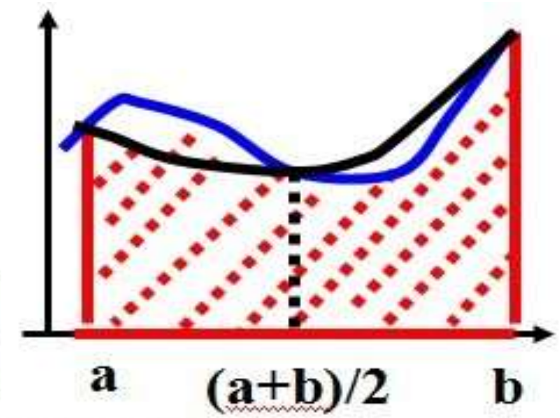
5 辛普森算法

01 一元辛普森

原理

Simpson公式

$$\int_a^b f(x)dx \approx \frac{1}{6}(b-a)\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$



Simpson公式是以函数f(x)在a, b, (a+b)/2这三点的函数值f(a), f(b), $f(\frac{a+b}{2})$ 的加权平均值 $\frac{1}{6}(f(a)+4f(\frac{a+b}{2})+f(b))$ 作为平均高度f(ξ)的近似值而获得的一种数值积分方法。

代码

```
for( i = 1 ; i <= ( up - down ) / minizone ; i ++ ){
        delta =  ((right - left ) / 6.0)*( lastsolve( orifunc , left ) + 4.0 *
lastsolve( orifunc , ( left+right) / 2.0 ) + lastsolve( orifunc , right ) );
        sum = sum + delta ;
        left = right ;
        right = left + minizone ;
```

02

二元辛普森

$$\int_a^b\int_c^d f(x,y)dxdy = \int_a^b F_x(y)dy \approx \frac{b-a}{6}(F_x(a)+4F_x(\frac{a+b}{2})+F_x(b))$$

$$F_x(y)=\int_c^d f(x,y)dx \approx \frac{d-c}{6}(f(c,y)+4f(\frac{d+c}{2},y)+f(d,y))$$

中心思想

根据辛普森公式固定y的值以后，对f(x,y)求关于x的积分，之后再对y求积分即可。

```
double exact_fy_binary( string orifunc , double a , double b , double y , double esp ){
  double S,L,R,c,answer;
  c=(a+b)/2.0;
  S=fixySimpson_binary( orifunc , a , b , y );
  L=fixySimpson_binary( orifunc , a , c , y );
  R=fixySimpson_binary( orifunc , c , b , y );
if(abs(L+R-S)<=15.0*esp)//判断是否满足精度
    answer=L+R+(L+R-S)/15.0;
    else
answer=exact_fy_binary(orifunc,a,c,y,esp/2.0)+exact_fy_binary(orifunc,c,b,y,esp/2.0);
  return answer;
}
```

此程序中判断是否达到精度，如果达到精度则输出L+R+(L+R-S)/15.0，如果没达到精度，则将积分区间平均划分为两个区间，每一个区间需要达到的精度为原理精度的1/2，区间划分得越多计算所得积分的偏差也就越小。

```
//固定y的值,计算下限为a，上限为b的x的积分
double fixySimpson_binary( string orifunc , double a , double b , double y ){
  double c,answer;
  c =(a+b)/2.0;//求得a,b的平均值c
  answer=(binarysolve(orifunc,a,y)+4*binarysolve(orifunc,c,y)+binarysolve(orifunc,b,y))*(b-a)/6.0;
    return answer;
}
```

此段程序就是将y固定为一个准确的数字，利用一元辛普森公式计算固定的y值下x的积分。此函数中调用了之前编写的一元辛普森法计算积分的函数binarysolve。

02 二元辛普森

$$\int_a^b\int_c^d f(x,y)dxdy = \int_a^b F_x(y)dy \approx \frac{b-a}{6}(F_x(a)+4F_x(\frac{a+b}{2})+F_x(b))$$

$$F_x(y)=\int_c^d f(x,y)dx \approx \frac{d-c}{6}(f(c,y)+4f(\frac{d+c}{2},y)+f(d,y))$$

```
double Simpson_binary( string orifunc , double downx , double upx , double downy , double upy , double esp ){
    double midy,answer;
    midy=(upy+downy)/2;
    answer=(exact_fy_binary( orifunc , downx , upx , downy , esp )+4*exact_fy_binary( orifunc , downx , upx , midy , esp )+exact_fy_binary( orifunc , downx , upx , upy , esp ))*(upy-downy)/6;
    return answer;
}
```

将固定y值下的x的积分计算出来以后，我们只需要再对y进行积分即可得出二元函数的二重积分，Simpson_binary就是在exact_fy_binary的基础上写出的函数，对y积分我们只需再用一次一元辛普森公式即可得出最后的结果。

02

二元辛普森

$$\int_a^b \int_c^d f(x,y)\,dx\,dy = \int_a^b F_x(y)\,dy \approx \frac{b-a}{6}\left(F_x(a) + 4F_x\left(\frac{a+b}{2}\right) + F_x(b)\right)$$

$$F_x(y) = \int_c^d f(x,y)\,dx \approx \frac{d-c}{6}\left(f(c,y) + 4f\left(\frac{d+c}{2}, y\right) + f(d,y)\right)$$

```
double exact_Simpson_binary( string orifunc , double downx , double upx , double downy , double upy , double esp ){
    double S,L,R,midy,answer;
    midy=(upy+downy)/2.0;
    S=Simpson_binary( orifunc , downx , upx , downy , upy , esp );
    L=Simpson_binary( orifunc , downx , upx , downy , midy , esp );
    R=Simpson_binary( orifunc , downx , upx , midy , upy , esp );
    if(abs(L+R-S)<=15.0*esp)//判断是否满足精度
        answer=L+R+(L+R-S)/15.0;
        else answer=exact_Simpson_binary( orifunc , downx , upx , downy , midy , esp/2.0 )+exact_Simpson_binary( orifunc , downx , upx , midy , upy , esp/2.0 );
    return answer;}
```

接下来与exact_fy_binary函数的思想一致，即是为了算出更为精确的二重积分，在main函数中我们将最后的精度控制为0.00001，故而用此法得出的积分值将会十分准确。

6 蒙特卡洛投点法

- 数学法基础
- 代码讲解
- 重难点问题

蒙特卡洛投点法数学基础

蒙特卡洛数学基础：概率论伯努利大数定律。

定理（**伯努利大数定理**）

设 n_A 是 n 次独立重复试验中事件 A 发生的次数，p 是事件 A 在每次试验中发生的概率，则对于任意正数 $\varepsilon > 0$，有

$$\lim_{n\to\infty} P\left\{\left|\frac{n_A}{n} - p\right| < \varepsilon\right\} = 1 \text{ 或 } \lim_{n\to\infty} P\left\{\left|\frac{n_A}{n} - p\right| \geq \varepsilon\right\} = 0.$$

蒙特卡洛投点法计算定积分：设(X,Y)服从正方形{0≤x≤1,0≤y≤1}上的均匀分布，则可知 X,Y分别服从 [0,1] 上的均匀分布，且X,Y相互独立。记事件A={Y≤f(X)}，则A的概率为P(A)=P(Y≤f(X))，即定积分J的值 就是事件A出现的频率。
同时，由伯努利大数定律，我们可以用重复试验中A出现的频率作为p的估计值。即将(X,Y)看成是正方形{0≤x≤1,0≤y≤1}内的随机投点，这种方法就叫随机投点法。用随机点落在区域y≤f(x)中的频率作为定积分的近似值。

蒙特卡洛投点法数学基础

0-1上的积分

1）先用计算机产生在(0,1)上均匀分布的2n个随机数，组成n对随机数（x_i，y_i）,i=1,2,3,4......n,这里n可以很大，譬如n=10^4，甚至n=10^5.

2）对n对数据（x_i，y_i）,i=1,2,3,4......n,记录满足如下不等式$y_i \leq$ f(x_i)的次数。这就是事件A发生的频数s_n,由此可得事件A发生的频率$\frac{s_n}{n}$，又由于整个面积为1，所以J=$\frac{s_n}{n}$

非0-1上的积分

1)对于一般区间【a,b】上的定积分$J'=\int_a^b g(x)\,dx$,做线性变换y=（x-a）/（b-a），即可化为[0,1]区间上的积分。

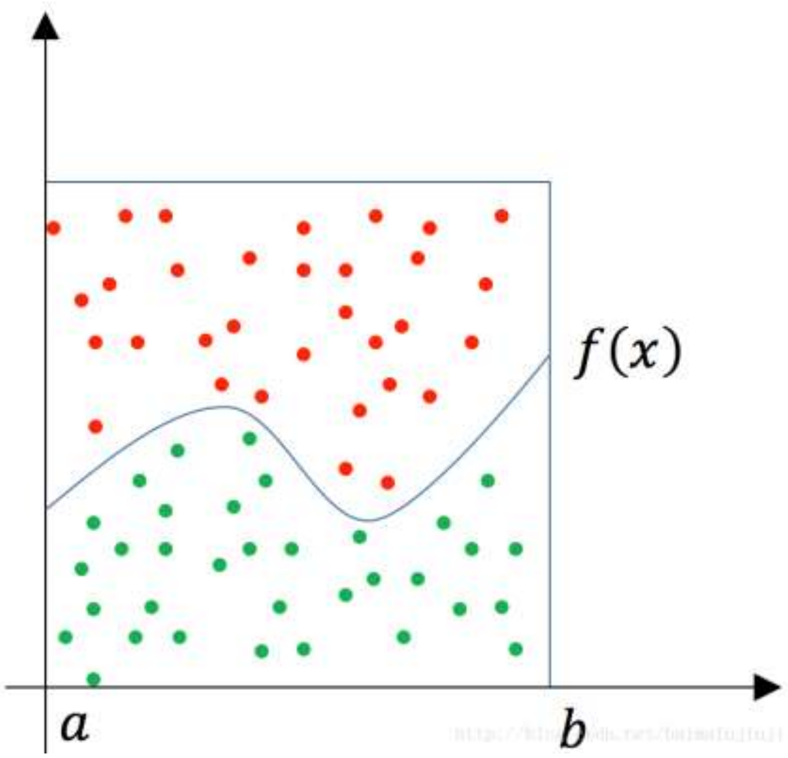
2)进一步若c≤ g(x) ≤d，可令f(y)=$\frac{1}{d-c}$[g(a+(b-a)y)-c)],则0≤ f(y) ≤1.

3)此时有$J'=\int_a^b g(x)\,dx$=$S_0\int_0^1 f(y)$dy+c(b-a)

其中S_0=(b-a)(d-c).这说明以上用蒙特卡罗方法计算定积分带有普遍意义.

蒙特卡洛投点法数学基础

蒙特卡洛投点法：如下图所示，有一个函数f(x)，若要求它从a到b的定积分，其实就是求曲线下方的面积。这时我们可以用一个比较容易算得面积的矩型罩在函数的积分区间上（假设其面积为Area）。然后随机地向这个矩形框里面投点，其中落在函数f(x)下方的点为绿色，其它点为红色然后统计绿色点的数量占所有点（红色+绿色）数量的比例为r，那么就可以据此估算出函数f(x)从a到b的定积分为Area×r



蒙特卡洛投点法求一元定积分思维导图

输入积分上下极限n,m

确定可以包含函数的矩形区域s={（x，y）|m<x<n,Min0<y<Max0 }

产生矩形内的随机点（x,y)

若随机点位于x轴上方

找到落在区间S1={（x，y）|m<x<n,0<y<Max0}中的点的个数enter_up和落在区间S2={（x，y）|m<x<n，0<y<f(x)}中的点的个数include_up；

统计include_up占enter_up的比例r1

面积计算Area1 = (n - m) * Max0 * r1

若随机点处于x轴下方

找到落在区间S3={（x，y ）|m<x<n,Min0<y<0}中的点的个数 enter_down和落在区间S4={（x，y ）|m<x<n，f(x)<y<0}中的点的个数include_down；

统计include_down占enter_down的比例r2

面积计算Area2 = (n - m) * (-Min0) * r2

定积分的计算值Area1+Area2

蒙特卡洛投点法代码讲解

定义相关变量

```
double Montecarlo( string orifunc , double down , double
up )
{
    double m = down ;
     double n = up ;
      double Max_point = m ;
     double Min_point = m ;
     double Max0 = lastsolve( orifunc, Max_point ) ;
     double Min0 = lastsolve( orifunc, Min_point ) ;
     double delta = 0.001;
```

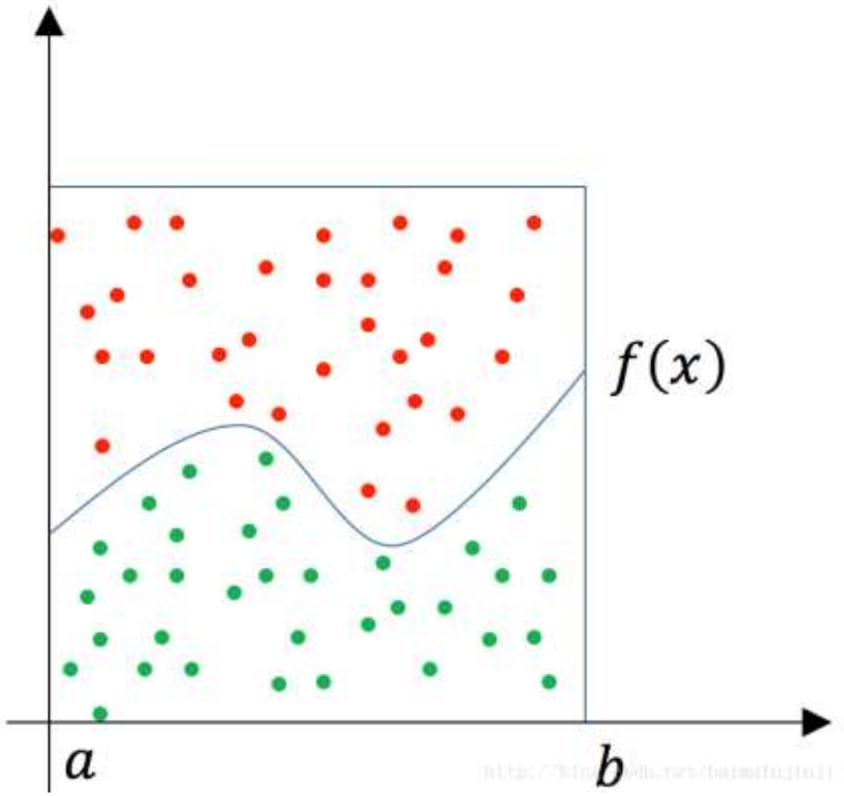
定义了一个函数 Montecarlo，确定了一个被积函数，并定义了它的上下极限n，m。

确定分割精度Δ = 0.001

找到函数最大值与最小值，确定一个包含被积函数的矩形S={（x，y）|m<x<n,Min0<y<Max0}

```
for ( i = 0 ; i < ( up - down ) / delta ; i = i + 1 ) {
    if ( lastsolve ( orifunc , Min_point + delta ) < Min0 ) {
        Min0 = lastsolve ( orifunc , Min_point + delta ) ;
        Min_point = Min_point + delta ;
    }
}
for( i = 0 ; i < ( up - down ) / delta ; i = i + 1 ) {
    if ( lastsolve ( orifunc , Max_point + delta ) > Max0) {
        Max0 = lastsolve ( orifunc , Max_point + delta ) ;
        Max_point = Max_point + delta ;
    }
}
```

下面以寻找最小值为例进行说明。利用for循环，循环次数为（up - down) / delta。从函数的下限开始循环，首先初始函数的最小值点为m，然后初始最小值点m对应的函数值为最小值 Min0 = lastsolve(orifunc, Min_point)，然后令最小值点为m+delta，如果m+delta点对应的函数值小于m点对应的函数值，则令最小值点为m+delta，最小值为Min0 = lastsolve (orifunc , Min_point + delta)，否则，最小值不更改，然后依次把最小值增加delta，进行循环寻找，直到循环到上极限n处。这样就找到了函数的最小值Min0 。同理我们可以找到函数的最大值Max0。



```
 if ( Max0 >= Min0 && Min0 >
0 ) {
        Min0 = 0 ;
   }
   if ( Max0 >= Min0 && Max0
< 0 ) {
        Max0 = 0 ;
   }
```

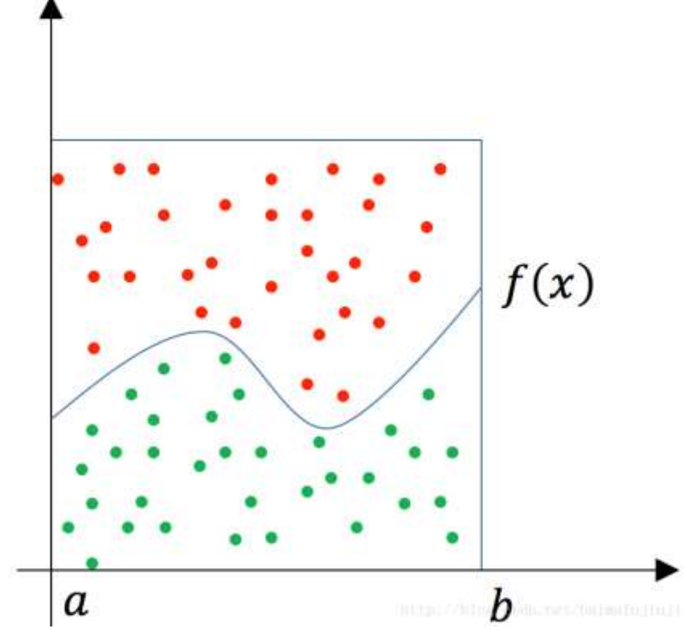
当最大值大于最小值且最小值大于零的情况下，投点区域的下界应该为x轴，令最小值为0；如果最大值大于最小值且最大值小于0的情况下投点区域的上界应该为x轴，令最大值为0。这样就确定了函数的最大值最小值。进而确定了一个可以包含函数的矩形S={（x，y）|m<x<n,Min0<y<Max0}。

产生随机数

```
default_random_engine e ( time(0) ) ;
uniform_real_distribution<double> u ( m , n ) ;
srand( (double)time(NULL) ) ;
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
    x = u(e);
    y = ( rand() / (double)RAND_MAX ) * ( Max0 - Min0 ) + Min0 ;
```

用时间序列产生随机数，由于在同一时间节点产生的随机数是一样的，在这里用两种方法产生随机数，就避免了产生一样随机数的问题。

确定绿点占总点数的比例



```
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
        x = u(e);
  y = ( rand() / (double)RAND_MAX ) * ( Max0 - Min0 ) +
Min0 ;
    if ( y >= 0 ) {
      enter_up = enter_up + 1 ;
      if ( y <= lastsolve( orifunc , x ) ) {
        include_up = include_up + 1 ;
      }
    }
    if ( y < 0 ) {
      enter_down = enter_down + 1 ;
      if( lastsolve ( orifunc , x ) >= y ) {
        include_down = include_down + 1 ;
      }
    }
  }
    r1 = include_up / enter_up ;
    r2 = include_down / enter_down ;
```

此段代码用于确定找到落在区间S1={（x，y）|0≤y≤max0，m≤x≤n}中的点的个数enter_up和落在区间S2={（x，y）|0≤y≤f（x），m≤x≤n}中的点的个数include_up；和找到落在区间S3={（x，y）|min0≤y≤0，m≤x≤n}中的点的个数enter_down和落在区间S4={（x，y）|f（x）≤y≤0 , a≤x≤b}中的点的个数include_down；并确定统计S2占S1的比例r1，统计S4占S3的比例r2。

确定面积

```
if ( include_up == 0 ) {
    Area1 = 0 ;
}
else {
    Area1 = (n - m) * Max0 *  r1 ;
}
if( include_down == 0 ) {
    Area2 = 0 ;
}
else {
    Area2 = (n - m) * (-Min0) * r2 ;
}
return Area1 + Area2;
```

本段代码用于求面积，把函数分为在x轴上下两部分。
当include_up == 0 时，说明产生的所有随机数对都在x轴下方，此时计算面积 Area1 = 0 ;
当include_up =！ 0 时 Area1 = (n - m) * Max0 * r1 ;同理可求Area2.
最后所求定积分的值为Area1+Area2.

蒙特卡洛投点法求二元定积分思维导图

- 输入关于x的积分上下极限n,m，输入关于y的积分上下极限n1,m1
- 确定可以包含函数的长方体区域v={（x，y，z）|m<x<n,m1<y<n1,Min0<z<Max0 }
  - 产生长方体内的随机点（x,y,z)
    - 若随机点位于x0y平面上方
      - 找到落在区间V1={（x，y，z）|m<x<n,m1<y<n1,0<z<Max0}中的点的个数enter_up和落在区间V2={（x，y，z）|m<x<n,m1<y<n1,0<z<f(x,y)}中的点的个数include_up；
        - 统计include_up占enter_up的比例r1
          - 面积计算Area1 =(n1 - m1) * (n - m) * Max0 * r1
    - 若随机点处于x0y平面下方
      - 找到落在区间V3={（x，y ,z）|m<x<n,m1<y<n1,Min0<z<0}中的点的个数 enter_down和落在区间V4={（x，y，z）|m<x<a,m1<y<n1,f(x,y,z)<z<0}中的点的个数include_down；
        - 统计include_down占enter_down的比例r2
          - 面积计算Area2 = (n1 - m1) * (n - m) * (-Min0) * r2

定积分的计算值Area1+Area2

重 难 点 ： 随 机 投 点 的 产 生

在利用蒙特卡洛投点法求定积分时，需要用到随机数的产生。在用随机数的产生时我遇到了重大的问题。

第一个问题，在利用rand()和srand()产生随机数时，

```
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
x =  ( rand() % ( n- m + 1) )+ m ;
y = ( rand() % ( Max0 - Min0 + 1) )+ Min0 ;
```

希望每次循环都可以产生[m,n]和[Min0，Max0]之间的两个不同的随机数，但是每次循环都会产生两个一样的随机数对即x = y。

第二个问题:

每次产生的随机数都是整数，但我们希望可以产生随机小数。

问题解决

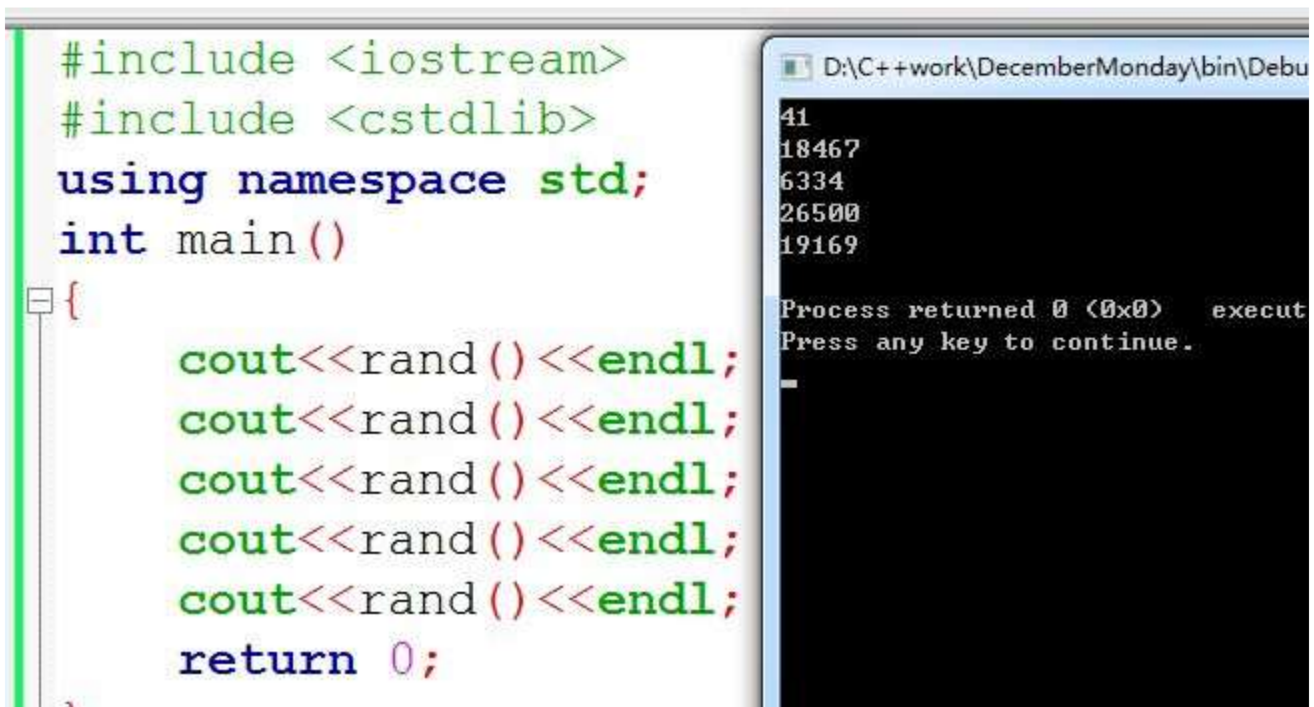
为了解决遇到的问题，首先需要了解随机数的产生。随机数的产生有两种。

第一种：

C++中没有自带的random函数，要实现随机数的生成需要使用rand()和srand()。

rand()

rand()会返回一随机数值，范围在0至MAND_MAX间。MAND_MAX定义在stdid.h,其值为2147483647.



```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    cout<<rand()<<endl;
    cout<<rand()<<endl;
    cout<<rand()<<endl;
    cout<<rand()<<endl;
    cout<<rand()<<endl;
    return 0;
}
```

```
D:\C++work\DecemberMonday\bin\Debu
41
18467
6334
26500
19169

Process returned 0 (0x0)   execut
Press any key to continue.
```

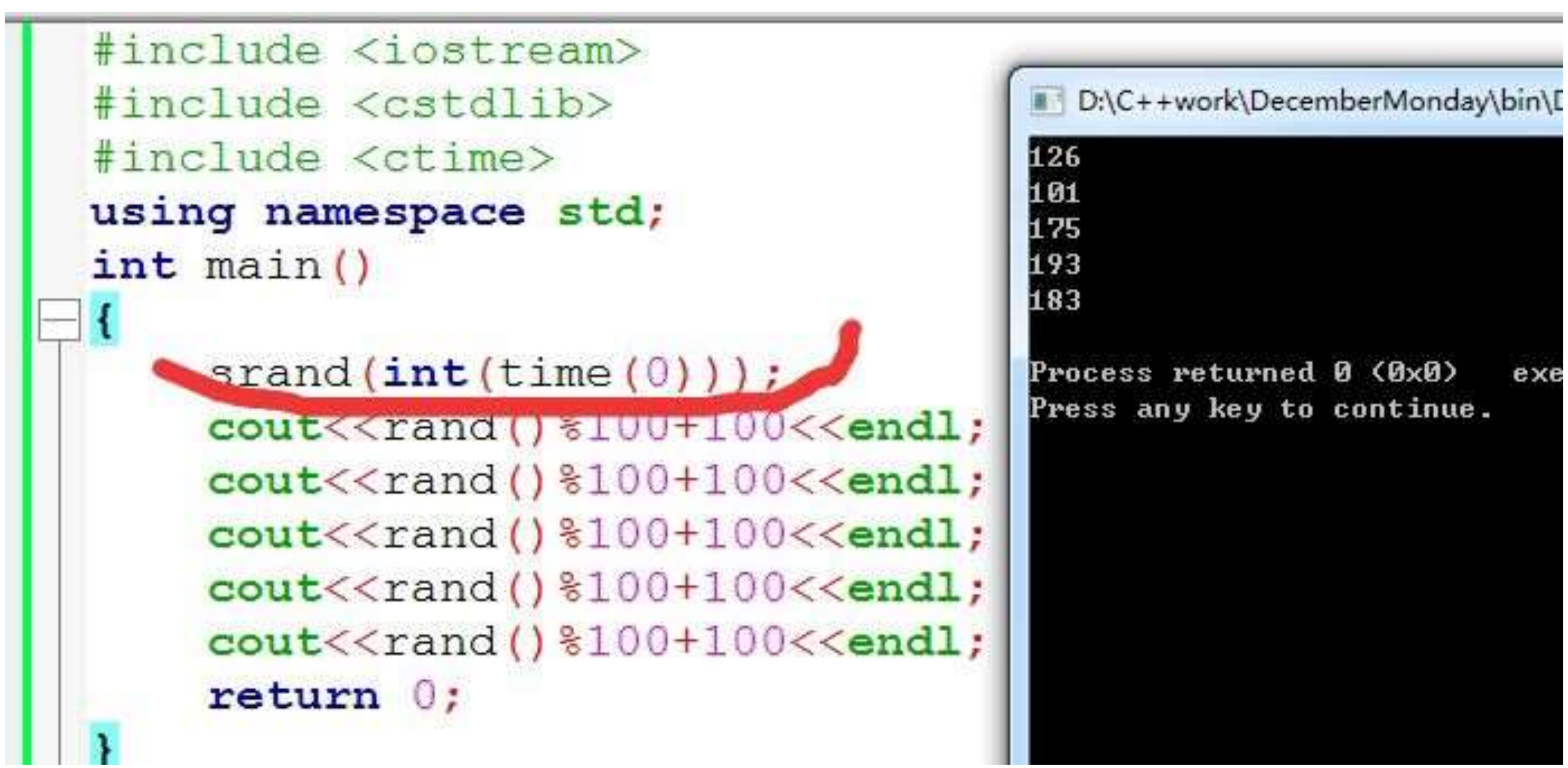
但是因为没有随机种子所以，下一次运行也是这个数，因此就要引出srand().

srand()

srand()可用来设置rand()产生随机数时的随机种子。通过设置不同的种子，我们可以获取不同的随机序列。

可以利用srand((int)(time(NULL))的方法，利用系统时钟，产生不同的随机数种子，不过需要调用time(),需要加入头文件<ctime>。

time（0）可以输出一个与时间有关的数。



获取[m,n]之间的随机整数使用（rand() %（n- m + 1))+ m；

获取0~1之间的浮点数使用 rand()/double(MAND_MAX）；

利用0~1之间的浮点数使用rand()/double(MAND_MAX)，可以解决只能产生随机整数的问题。
首先产生0~1内浮点数然后在扩展到固定范围内。

解决方法：

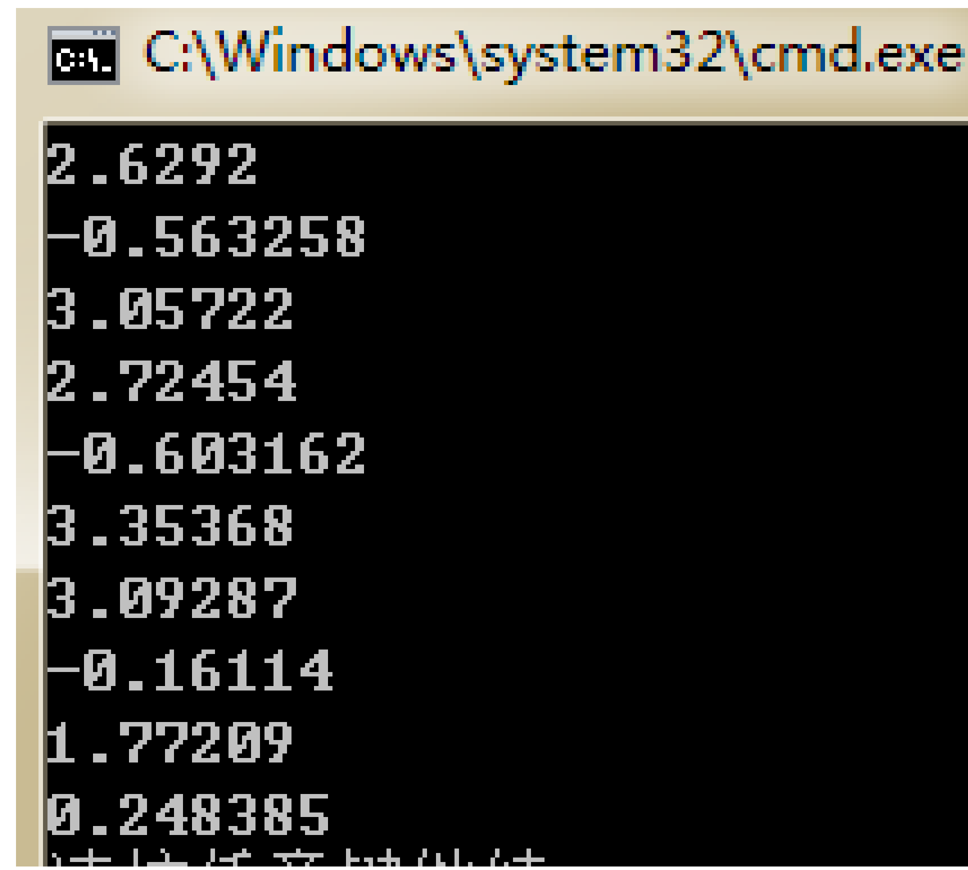
```
srand( (double)time(NULL) ) ;
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
x = ( rand() / (double)RAND_MAX ) * ( n - m ) + m ;
y = ( rand() / (double)RAND_MAX ) * ( Max0 - Min0 ) + Min0 ;
```

这样就可以产生随机小数。

```
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
x = ( rand() % ( n- m + 1) )+ m ;
y = ( rand() % ( Max0 - Min0 + 1) )+ Min0 ;
```

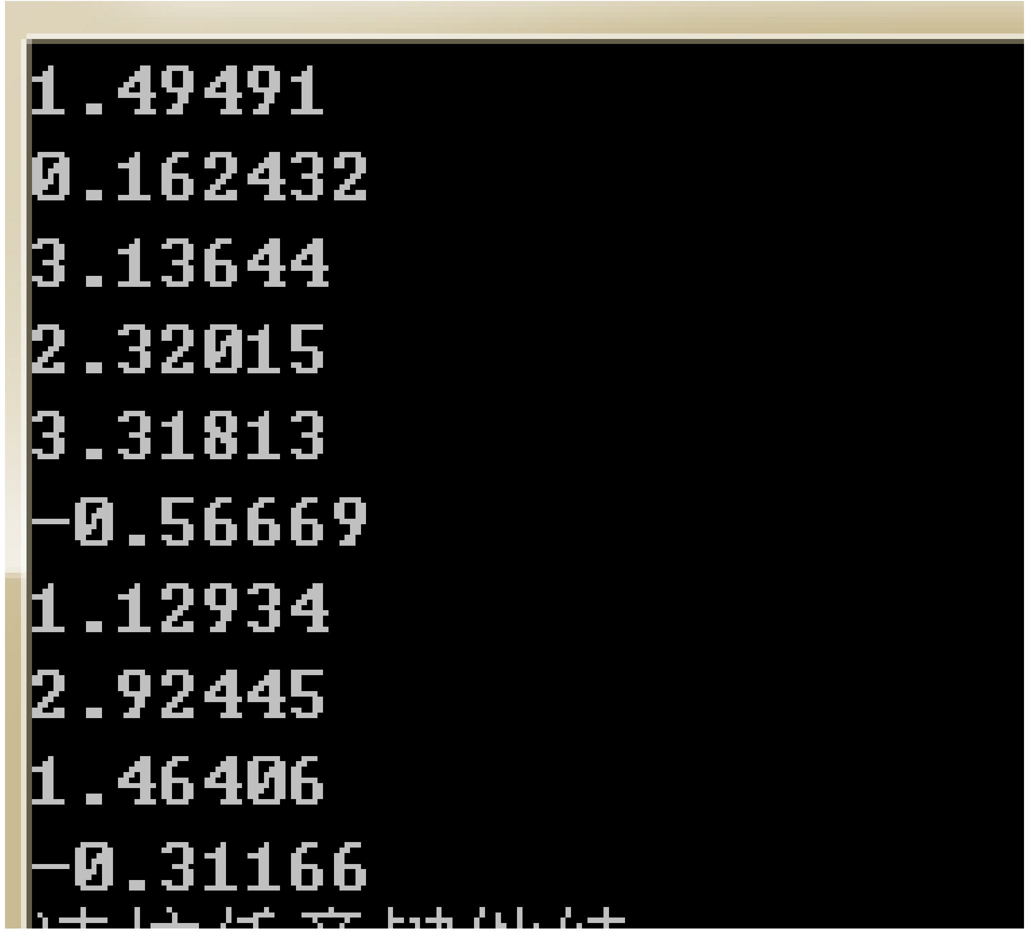
第二种方法：
C++新标准，有一个叫随机数引擎的东西。
包含于头文件 <random>。
随机数库由：引擎，分布组成。

```
#include<random>
#include<iostream>
using namespace std;

int main()
{
	default_random_engine e;
	uniform_real_distribution<double> u(-1.2,3.5);
	for(int i = 0; i < 10; ++i)
		cout << u(e) << endl;
	return 0;
}
```



如果多次运行，会发现结果是一样的。
所以需要设置种子。

```
#include<random>
#include<iostream>
#include<ctime>
using namespace std;

int main()
{
	default_random_engine
e(time(0));
	uniform_real_distribution<double
> u(-1.2,3.5);
	for(int i = 0; i < 10; ++i)
		cout << u(e) << endl;
	return 0;
}
```



为了产生两一次循环可以产生两个不同的随机数，利用下面的代码。

```
default_random_engine e ( time(0) ) ;
        uniform_real_distribution<double> u ( m , n ) ;
srand( (double)time(NULL) ) ;
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
x = u(e);
y = ( rand() / (double)RAND_MAX ) * ( Max0 - Min0 ) + Min0 ;
```

同时使用，两种产生随机数的方法就可以得到两个不同的随机数。

```
srand( (double)time(NULL) ) ;
for ( i = 0 ; i < ( up - down ) / delta ; i++ ) {
x = ( rand() / (double)RAND_MAX ) * ( n - m ) + m ;
y = ( rand() / (double)RAND_MAX ) * ( Max0 - Min0 ) + Min0 ;
```

7 蒙特卡洛平均值法

- 数学法基础
- 代码讲解

蒙特卡洛平均值法
数学基础

直观解释：

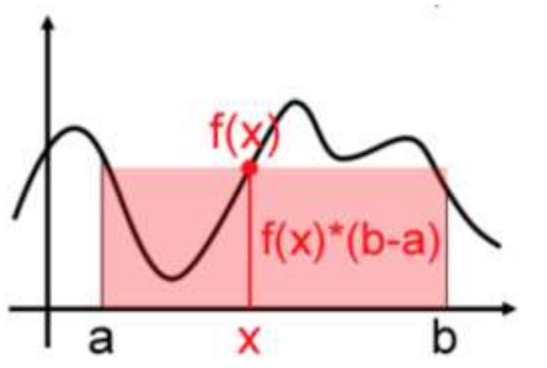


Figure 2: the curve can be evaluated at x and the result can be multiplied by (b - a). This defines a rectangle which can be seen as a very crude approximation of the integral.

当我们在[a,b]之间随机取一点xx时，它对应的函数值就是f(x)，然后变可以用f(x)×(b−a)来粗略估计曲线下方的面积（也就是积分），当然这种估计（或近似）是非常粗略的。

下面介绍一下利用蒙特卡洛法求定积分的第二种方法——期望法，有时也成为平均值法。

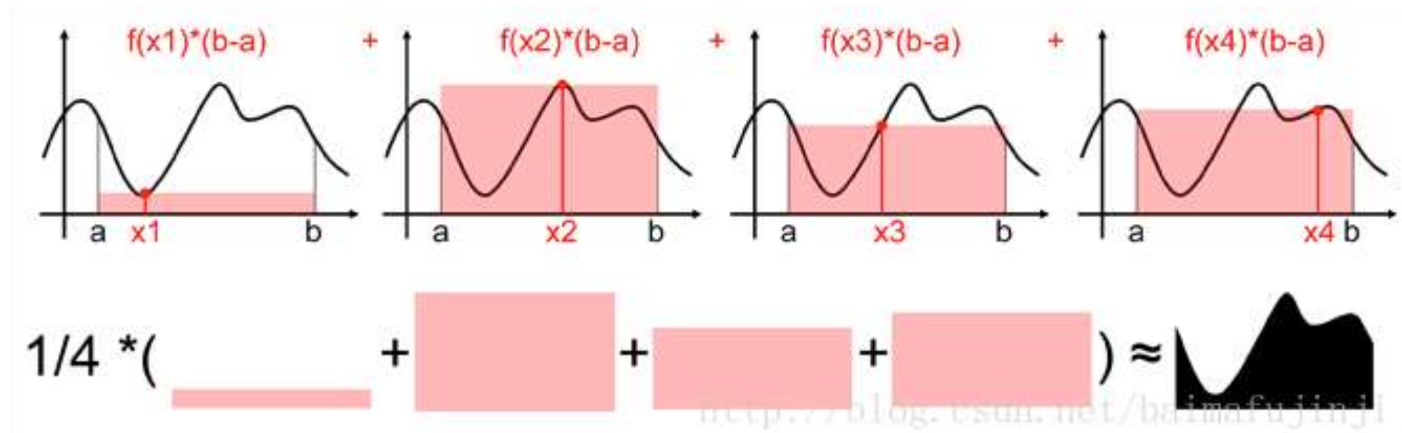
具体步骤如下：

产生[a,b]上的均匀分布随机变量Xi (i=1,2,⋯,N);

计算均值

$\bar{I}=b-a/N\sum g(X_i)$

并用它作为I的近似值，即$I\approx\bar{I}$。



于是我们想到在[a,b]之间随机取一系列点xi时（xi满足均匀分布），然后把估算出来的面积取平均来作为积分估计的一个更好的近似值。可以想象，如果这样的采样点越来越多，那么对于这个积分的估计也就越来越接。

蒙特卡洛平均值法求一元定积分代码讲解

定义相关变量，产生随机数

```
double montecarlo( string orifunc , double
down , double up )
{
 double m = down ;
double n = up ;
double delta = 0.001;
double y ;
 double x ;
   double value1 = 0 ;
   double value2 = 0

default_random_engine e ( time(0) ) ;
            uniform_real_distribution<double> u
( m , n ) ;
            for ( i = 0 ; i < (n - m)/delta ; i++ ) {
            x = u(e);
            y = lastsolve( orifunc , u(e) ) ;
```

定义一个新的函数montecarlo，定义积分上下限n，m。确定分割精度delta = 0.001。

用时间序列产生随机数。由于在同一个时间节点上产生的随机数是一样的，所以采用两种不同的方法，在同一时间节点上会产生两个不同的数。采用循环的形式产生足够多的想要的随机数。

计算面积求和得定积分

```
for ( i = 0 ; i < (n - m)/delta ; i++ ) {
        x = u(e);
        y = lastsolve( orifunc ,
u(e) ) ;
if ( y >= 0 ) {
        value1= value1 + y * ( n -
m ) ;
        }
        if ( y < 0 ) {
        value2 = value2 + ( - y ) *
( n - m ) ;
        }
return  value1 + value2  / (n -
m)/delta ;
```

如果产生的随机数在x轴上面，计算面积value1= y * (n - m)；如果产生的随机数在x轴下面，计算面积value2 = (- y) * (n - m)；在利用for循环，取(n - m)/delta 个不同的x点对应的y的值，计算面积随机数在x轴上面value1= value1 + y * (n - m)；随机数在x轴下面value2 = value2 + (- y) * (n - m)。最后用蒙特卡洛平均值求得的一元积分值为：value1 + value2 / (n - m)/delta。

蒙特卡洛平均值法求二元定积分代码讲解

```
void montecarlo_binary(string orifunc,double
downx,double upx,double downy,double upy)
{
    double m = downx ;
    double n = upx ;
    double m1 = downy ;
    double n1 = upy ;
    double x ,y ,z ;
    double delta = 0.001;
    int i ;
    double value1 = value2 = 0 ;
     double f ;
if ( ( upx - downx ) / delta  > ( upy -
downy ) / delta ) {
    f = ( upx - downx ) / delta ;
        }
        else {
      f = ( upy - downy ) / delta ;
        }
    default_random_engine e ( time(0) ) ;
            uniform_real_distribution<double> u
( m , n ) ;
    srand ( (double)time(NULL) ) ;
            for ( i = 0 ; i < f ; i++ ) {
            x = u(e);
            y = ( rand() / (double)RAND_MAX ) *
( upy - downy ) + downy ;
            z =  binarysolve ( orifunc , x , y ) ;
            if ( z >= 0 ) {
            value1 = value1 + z * ( n - m ) * ( n1 -
m1 ) ;
            }
            if ( z < 0 ) {
            value2 = value2 + ( - z ) * ( n - m ) *
( n1 - m1 ) ;
            }
            }
                    return  value1 + value2;
}
```

8 其他部分

日志文件

```
 ofstream fout;
    string path = "C:\\Users\\HP\\Desktop\\output.txt";
    fout.open(path,ios::app);
fout.close();
```

时间显示

```
SYSTEMTIME time
GetLocalTime(&time)
cout  << setfill('0') << setw(2) << time.wHour << "时"
         << setfill('0') << setw(2) << time.wMinute << "分"
         << setfill('0') << setw(2) << time.wSecond << "秒”
```

颜色控制

```
system("color F0");
```

总 结 部 分

先列好思路，划分模块，再进行具体操作。

多沟通，多交流，多进行思维碰撞，团结协作。

分工前先明确格式、变量名等内容，否则后期修改统一工程量十分巨大。

跨学科学习，多学科交叉融合

谢谢大家！！！