.....

★1-1.模拟追逐问题

在二维情况下，写出导弹A追逐导弹B时的位置、速度迭代计算公式（采用Euler算法）。设导弹A位置为(xA,yA)，速度大小恒为vA，方向始终对准导弹B；导弹B的位置为(xB,yB)，速度为(vxB,vyB)，其加速度(axB,ayB)为常矢量。

$$
\begin{aligned}
dx &= x_B - y_A \\
dy &= y_B - xAn \\
dis &= sqrt(dx*dx + dy*dy)//\text{计算}A\text{到}B\text{的距离} \\
v_{xAn+1} &= v_{An} * dx/dis \\
v_{yAn+1} &= v_{An} * dy/dis; //\text{计算}A\text{的速度分量} \\
x_{An+1} &= x_{An} + v_{xAn} * dt; \\
y_{An+1} &= y_{An} + v_{yAn} * dt; //\text{迭代计算}A\text{的位置} \\
x_{Bn+1} &= x_{Bn} + v_{xBn} * dt; \\
y_{Bn+1} &= y_{bn} + v_{yBn} * dt; //\text{迭代计算}B\text{的位置} \\
v_{xBn+1} &= v_{xBn} + a_{xBn} * dt; \\
v_{yBn+1} &= v_{yBn} + a_{yBn} * dt; //\text{迭代计算}B\text{的速度}
\end{aligned}
$$

★1-2.Euler算法的稳定性

给定一阶微分方程：$\frac{dy}{dt} = -py$（p是正的常数），写出其Euler算法数值解的迭代公式和稳定性条件。 Euler算法迭代公式：$y_{n+1} = y_n - py(n)\Delta t = (1 - p\Delta t)y_n$；稳定性条件：|1-p*Δt|<1，即Δt<2/p。

稳定性的推导过程：

- 由一阶微分方程可知，它的斜率是负的，y_{n+1}比y_n小。
- 因此，$y_{n+1} = (1 - p\Delta t)y_n$中的$|(1 - p\Delta t)| < 1$才会满足$y_{n+1}$比$y_n$小。

★2-1.Runge-Kutta算法

给定一阶微分方程：$\frac{dy}{dt} = at - by$（a、b是正的常数），写出其Runge-Kutta算法数值解的迭代公式，并分析其误差估算方法。

其中，Runge-Kutta算法迭代公式：

$$k_1 = f(y_n, t_n)$$
$$k_2 = f(y_n + \frac{k_1 dt}{2}, t_n + \frac{dt}{2})$$
$$k_3 = f(y_n + \frac{k_2 dt}{2}, t_n + \frac{dt}{2})$$
$$k_4 = f(y_n + k_1 dt, t_n + dt)$$
$$y_{n+1} = y_n + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

误差估算：Runge-Kutta算法具有四阶精度，即全局误差En正比于$(\Delta t)^4$。设Δt=0.2时，t从0迭代到tmax时，y的估值为y1；而Δt=0.1时，t从0迭代到tmax时，y的估值为y2；设y的真实值为ya，则$\frac{ya - y1}{ya - y2} \approx (\frac{0.2}{0.1})^4 = 16$，可得$\frac{y2 - y1}{ya - y2} \approx 15$，所以全局误差$E2 = ya - y2 \approx \frac{y2 - y1}{15}$，由此也可以估算出真实值$ya \approx \frac{16 \times y2 - y1}{15}$。

★2-2.给定一阶微分方程：$\frac{dy}{dt} = 5t^4$，初条件：t=0时，y=1。①写出Runge-Kutta算法迭代公式；②步长分别取Δt=4和Δt=2，计算t=4时y的估值y1和y2；③对于Δt=2，t=4时y的估值y2与真实值ya间的误差是多少？ Runge-Kutta算法迭代公式：

```
k1=f(tn);
k2=f(tn+Δt/2);
k3=f(tn+Δt/2);
k4=f(tn+Δt);
y(n+1)=y(n)+(k1+2*k2+2*k3+k4)*Δt/6;
```

取Δt=4，列表计算各值：

n	tn	yn	k1	k2	k3	k4
0	0	1	0	80	80	1280
1	4	$\frac{3203}{3}$				

取Δt=2，列表计算各值：

n	tn	yn	k1	k2	k3	k4
0	0	1	0	5	5	80
1	2	$\frac{103}{3}$	80	405	405	1280
2	4	$\frac{3083}{3}$				

Δt=2，t=4时的误差$E = ya - y2 \approx (y2 - y1)/15 = -8/3$

★3-1.梯形法一维数值积分

给定被积函数f(x)，积分区间[a,b]，写出其梯形法数值积分公式，并分析误差En与分割数n的关系。 梯形积分公式：$T=\frac{\Delta x}{2}[f(a)+f(b)+2\sum_{i=1}^{n-1}f(x_i)]$ 其中$\Delta x=\frac{b-a}{n}, x_i=a+\frac{b-a}{n}i$ 误差En正比于 $\frac{1}{n}^2$ 。设 $n_1=1024$ 时，积分估值为T_1，而 $n_2=2048$ 时，积分估值为 T_2，设积分真实值为 F_a，则有 $(F_a-T_1)/(F_a-T_2)\approx(\frac{n_2}{n_1})^2=4$，可得 $\frac{T_2-T_1}{F_a-T_2}\approx 3$，即误差 $E_2=F_a-T_2\approx\frac{T_2-T_1}{3}$，估算真实为 $F_a\approx\frac{4T_2-T_1}{3}$。

梯形法一维数值积分的算法实现。

给定被积函数f(x)=exp(-x)，用梯形法计算其在[0,1]区间的积分，输出结果和误差。

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double TrapezInt(double (*f)(double), double a, double b, double &e){
//梯形法数值积分，f指向一维被积函数，a、b为积分下限和上限，
//函数返回积分结果Tn，e用于记录最终结果与真实值间的误差估计e=Fa-Tn。
    double x,dx,dx2,sum,Tn,Tn0;
    unsigned long n=2,nmax=1<<20;//分割数
    dx2=b-a;
    dx=dx2/2.0;
    sum=0.5*((*f)(a)+(*f)(b));//先累加起点、终点的函数值
    Tn0=sum*dx2;//分割数为1的积分结果
    do{
        x=a+dx;//x1
        while(x<b){
            sum+=(*f)(x); //累加新的分割点函数值
            x+=dx2;//x=x+2*dx
        }
        Tn=sum*dx;
        e=Tn-Tn0;
        if((e<0?-e:e)<1e-8){//误差已足够低，中断计算
            break;
        }
        if(n>=nmax){//分割数已达到最大分割数，中断计算
            break;
        }
        if(kbhit())if(getch()==27)break;//用户按了Esc键，中断计算
        dx2=dx; dx/=2.0; n*=2; Tn0=Tn; //步长减半，分割数翻倍，记录积分结果，为下一趟更高精度的计算做
准备
    }while(1);
    e/=3.0; //估算误差e=Fa-Tn=(Tn-Tn0)/3
    return Tn;
}

double func(double x){  return exp(-x); }
int main(){
    double Tn,et;
    Tn=TrapezInt(func,0,1,et);
```

```
    printf("Tn=%.14lg, et=%.14lg\n",Tn,et);
    return 0;
}
```

看上去很长，但核心的部分并没那么复杂，去掉无关紧要的部分，可以写成下面的样子。虽然和上面的写法不同，但思路是一样的。

```
double TrapezInt((*f)(double),double a,double b){
    double dx2 = b-a;
    double dx = dx2/2.0;
    double sum_temp = (*f)(a)+(*f)(b);
    int n=2;
    int nmax=1<<20;
    do{
        x = a+dx;
        while(x<b){
            sum_temp = sum_temp + 2.0 * (*f)(x);
            x+=dx2;
        }
        sum = sum_temp * dx / 2.0;
        dx2 = dx;
        dx = dx / 2.0;
        n = n * 2;
    }while(n<nmax)
    return sum;
}
```

给定被积函数$f(x)=5x^4$，积分区间[0,4]，采用**梯形法**，分别计算分割数n=2和4时的积分估值，进一步计算n=4时积分真实值与估值间的误差。 列表计算各点函数值：

f(0)	f(1)	f(2)	f(3)	f(4)
0	5	80	405	1280

n=2时，$T2=(4-0)*\frac{\frac{1}{2}f(0)+f(2)+\frac{1}{2}f(4)}{2}=1440$； n=4时，

$T4=(4-0)*\frac{\frac{1}{2}f(0)+f(1)+f(2)+f(3)+\frac{1}{2}f(4)}{4}=1130$； n=4时的误差

$E_4=F_a-T_4\approx\frac{T_4-T_2}{3}=\frac{-310}{3}$，也就是真实值$F_a\approx T_4+E_4=\frac{3080}{3}$。

★3-2.辛普森法一维数值积分。

给定被积函数f(x)，积分区间[a,b]，写出其辛普森法数值积分公式，并分析误差En与分割数n的关系。 辛普森积分公式：

$$S=\frac{b-a}{n*3}[f(a)+f(b)+4\sum_{i=1,\delta i=2}^{n-1}f(x_i)+2\sum_{i=2,\delta i=2}^{n-2}f(x_i)]$$

或，

$$S=\frac{\Delta x}{3}[f(a)+f(b)+4\sum_{i=1,\delta i=2}^{n-1}f(x_i)+2\sum_{i=2,\delta i=2}^{n-2}f(x_i)]$$

其中$\Delta x=\frac{b-a}{n}, x_i=a+\frac{b-a}{n}i$, 误差$E_n$正比于$\frac{1}{n}^4$。设n=1024时，积分估值为$S_1$，而n=2048时，积分估值为$S_2$，设积分真实值为$F_a$，则有$\frac{F_a-S_1}{F_a-S_2}\approx 16$，可得$\frac{S_2-S_1}{F_a-S_2}\approx 15$，即误差$E_2=F_a-S_2\approx\frac{S_2-S_1}{15}$。对比梯形法、辛普森法积分公式可以发现，$S_2=\frac{4T_2-T_1}{3}$。

辛普森法一维数值积分的算法实现

给定被积函数f(x)=exp(-x)，用辛普森法计算其在[0,1]区间的积分，输出结果和误差。

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double SimpsonInt(double (*f)(double), double a, double b, double &e){
//辛普森法数值积分，f指向一维被积函数，a、b为积分下限和上限，
//函数返回积分结果Sn，e用于记录最终结果与真实值间的误差估计e=Fa-Sn。
    double x,dx,dx2,sum,sum0,Sn,Sn0;
    unsigned long n=4,nmax=1<<20;//分割数
    dx2=(b-a)/2.0;//2倍区间宽度
    dx=dx2/2.0;//分割数为4的dx
    sum=4.0*(*f)(a+dx2);
    sum0=(*f)(a)+(*f)(b)+sum;
    Sn0=sum0/3.0*dx2;//分割数为2的积分结果
    do{
        sum0-=sum/2.0;
        x=a+dx;
        sum=0.0;
        while(x<b){
            sum+=(*f)(x);
            x+=dx2;
        }
        sum*=4.0;
        sum0+=sum;
        Sn=sum0*dx/3.0; //Cotes公式：Cn=(16*Sn-Sn0)/15
        e=Sn-Sn0;
        if((e<0?-e:e)<1e-8){//误差已足够低，中断计算
            break;
        }
        if(n>=nmax){//分割数已达到最大分割数，中断计算
            break;
        }
        if(kbhit())if(getch()==27)break;//用户按了Esc键，中断计算
        dx2=dx; dx/=2.0; n*=2; Sn0=Sn; //步长减半，分割数翻倍，记录积分结果，为下一趟更高精度的计算做
准备
    }while(1);
    e/=15.0; //估算误差e=Fa-Sn=(Sn-Sn0)/15
    return Sn;
}
```

```
double func(double x){  return exp(-x); }
int main(){
    double Sn,es;
    Sn=SimpsonInt(func,0,1,es);
    printf("Sn=%.14lg, es=%.14lg\n",Sn,es);
    return 0;
}
```

就和梯形法一样，取出精化部分，就是下面的代码。

```
double SimpsonInt((*f)(double),double a,double b){
    double dx2 = (b-a) / 2.0;
    double dx = dx2 / 2.0;
    double temp = 4.0 * (*f)(a+dx2);
    double sum_temp = (*f)(a) + (*f)(b) + temp;
    double sum;
    int n = 2;
    int nmax = 1 << 20;
    double x;
    do{
        sum_temp -= (temp / 2.0);
        temp = 0;
        x = dx;
        while(x<b){
            temp += (*f)(x);
            x += dx2;
        }
        temp *= 4.0;
        sum_temp += temp;
        sum = sum_temp*dx/3.0;
        dx2 = dx;
        dx /= 2.0;
        n *= 2;
    }while(n<nmax)
    return sum;
}
```

给定被积函数$f(x)=5x^4$，积分区间[0,4]，采用**Simpson法**，分别计算分割数n=2和4时的积分估值，进一步计算n=4时积分真实值与估值间的误差。 列表计算各点函数值：

f(0)	f(1)	f(2)	f(3)	f(4)
0	5	80	405	1280

n=2时，$S_2=(4-0)\times\frac{f(0)+4f(2)+f(4)}{3\times 2}=\frac{3200}{3}$； n=4时，

$S4=(4-0)\times\frac{f(0)+4f(1)+2f(2)+4f(3)+f(4)}{3\times 4}=\frac{3080}{3}$； n=4时的误差

$E_4=F_a-S_4\approx\frac{S_4-S_2}{15}=\frac{-8}{3}$，也就是真实值$F_a\approx S_4+E_4=1024$。

★4-1.二项分布随机数。

①概率分布函数：$P(\xi=k;n,q)=C(n,k)q^k(1-q)^{n-k}=\frac{n!}{k!(n-k)!}q^k(1-q)^{n-k}$,其中$[0\le k\le n]$， ②生成方法（直接法）：

```
unsigned RndBin(unsigned n, double q){
//产生二项分布的随机整数。射击n发子弹，每发子弹命中概率为q，则命中k发的概率满足二项分布：
Pk(n,q)=C(n,k)*q^k*(1-q)^(n-k)，期望值【平均值】<k>=n*q，且k=n*q的概率最高。
    unsigned i,k;
    k=0;
    for(i=0;i<n;++i){
        if(rnd()<q)++k;
    }
    return k;
}
```

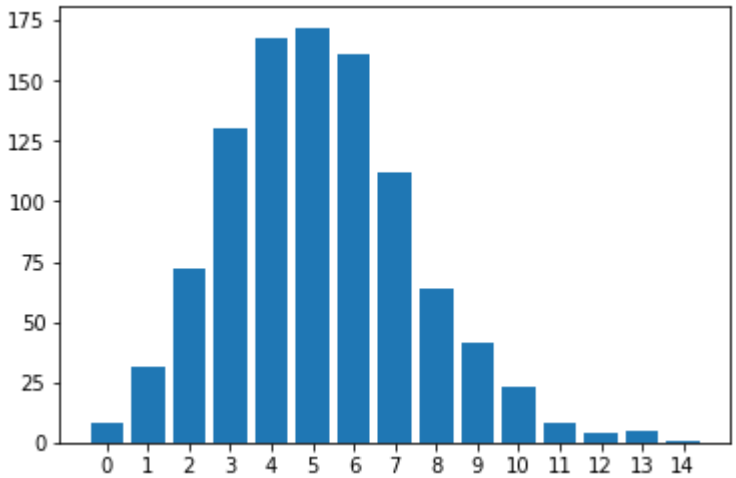
③举例：投掷一个色子6次，出现4次1点朝上的概率是多少？最有可能出现几次1点朝上？其概率是多少？（设色子掷出以后各面朝上的概率均平等）这里$n=6,q=1/6,k=4$，则
$P4(6,1/6)=C(6,4)*(1/6)^4*(5/6)^2=375/6^6=125/15552$。最有可能出现$n*q=1$次，其概率
$P1(6,1/6)=C(6,1)*(1/6)^1*(5/6)^5=(5/6)^5=3125/7776$。

★4-2.泊松分布随机数。

①概率分布函数：$P(\xi=k,\lambda)=\frac{\lambda^k}{k!}e^{-\lambda}[k\ge 0]$，它是二项分布的极限分布（n很大、q很小，λ=n*q）。 ②生成方法（直接法）：

```
unsigned RndPos(double lamda){
//产生泊松分布的随机整数。在相同的初条件下，一段时间内，发生核衰变的粒子数满足泊松分布：Pk(λ)=λ^k*exp(-λ)/k!，期望值【平均值】<k>=λ，且k=λ的概率最高。
    static double ez=exp(-lamda);
    double y;
    unsigned i;
    y=rnd();
    for(i=0;y>=ez;++i){
        y*=rnd();
    }
    return i;
}
```

上面的函数，每次运行都会生成一个整数字，这个数字的出现符合泊松分布。例如 lamda = 5时，并执行这个函数 100 次，会发现出现的数字



★4-3.均匀分布随机数。

①概率密度函数（已归一化∫(p(x),a:b)dx=1）：$p(\xi=x,a,b)=\frac{1}{b-a}(a\le x<b)$，②生成方法（反变换法）：分布函数$F(x)=\int_a^x p(x)dx=\frac{x-a}{b-a}=r\in[0,1)$均匀分布随机数，得到抽样公式：$x=r*(b-a)+a$。

```
double RndAver(double a, double b){
//产生[a,b)均匀分布的随机数：px(a,b)=1/(b-a)  (a<=x<b)
    return rnd()*(b-a)+a;
}
```

★4-4.指数分布随机数。

①概率密度函数（已归一化$\int_0^\infty p(x)dx=1$）：$p(\xi=x,\lambda)=\lambda e^{-\lambda x}(x\ge 0)$，②生成方法（反变换法）：分布函数$F(x)=\int_0^x p(x)dx=1-e^{-\lambda x}=r\in[0,1)$均匀分布随机数，所以$x=-\frac{ln(1-r)}{\lambda}$

```
double RndExp(double lamda){
//产生指数分布的随机数：px(λ)=λ*exp(-λ*x)  (x>=0)
    return -log(1-rnd())/lamda;
}
```

★4-5.正态分布随机数。

①概率密度函数：$p(\xi=x,u,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-u)^2}{2\sigma^2}}$，其中u为期望值（平均值），σ为单次测量的标准偏差，②生成方法（直接法）：$x=u+\sum_{i=1}^{12}r_i-6,r_i\in[0,1)$

```
double RndNorm(double u,double s){
//产生正态分布的随机数，u期望值（均值），s单次测量的标准偏差：px(u,σ)=exp(-(x-
u)^2/(2*σ^2))/(sqrt(2*π)*σ)
    int i;
    double sum,x;
    sum=0;
    for(i=0;i<12;++i)sum+=rnd();
    x=u+s*(sum-6);
    return x;
}
```

★4-6.混合同余法（线性同余法）

产生[0,1)均匀分布的随机数： $x=(ax+c)\%m, r=x\times\frac{1.0}{m}$

```
double rnd0(){
//混合同余法（线性同余法）产生[0,1)之间均匀分布的随机数
    static double m=217728.0,a=84589.0,c=45989.0,r,x=time(NULL);
    x=a*x+c;
    if(x>=m){x=x-m*(unsigned long)(x/m);}//求余
    r=x/m;//产生[0,1)之间的随机数
    return r;
}
```

★5-1.Monte-Carlo积分法之一：偶然命中法(hit or miss method)

描述算法步骤，约束条件。 约束条件：$0\le f(x)\le H$， 算法步骤：

```
double HitInt(double (*f)(double), double a, double b, double h, unsigned long n){
//偶然命中法进行一维数值积分：f指向被积函数，a、b为积分下限和上限，h为矩形高度【要求h>f(a~b)>0】，n为点
数。
    double x, y;
    unsigned long i,ns=0;
    for(i=1;i<=n;++i){ //执行n次抽样
        x=RndAver(a,b); //在[a,b)区间均匀地随机取xi
        y=RndAver(0,h); //在[0,h)区间均匀地随机取yi
        if(y<(*f)(x))++ns; //如果yi<f(xi)，则命中数ns加一
    }
    return h*(b-a)*ns/n; //返回数值积分结果Fn=h*(b-a)*ns/n
}
```

★5-2.Monte-Carlo积分法之二：平均抽样法(sample mean method)

描述算法步骤，并做误差分析。 算法步骤：

```
double MeanInt(double (*f)(double), double a, double b, unsigned long n, double &sigma){
```

```
//平均抽样法进行一维数值积分：f指向被积一元函数，a、b为积分下限和上限，n为抽样数，sigma记录单次抽样的标准偏差σ【平均值的标准偏差（标准误差）σm=σ/sqrt(n)】。
    double x, y, faver=0.0, f2aver=0.0;
    unsigned long i;
    for(i=0;i<n;++i){ //执行n次抽样
        x=RndAver(a,b);//在[a,b)区间平均抽样xi
        y=(*f)(x); //计算f(xi)
        faver+=y; //统计Σ(f(xi),1≤i≤n)
        f2aver+=y*y; //统计Σ(f^2(xi),1≤i≤n)
    }
    faver/=n; //计算f(x)的平均值<f>=Σ(f(xi),1≤i≤n)/n
    f2aver/=n; //计算f^2(x)的平均值<f^2>
    sigma=sqrt(f2aver-faver*faver); //计算单次测量的标准偏差σ=sqrt(<f^2>-<f>^2)
    return faver*(b-a); //返回数值积分结果Fn=<f>*(b-a)=(b-a)/n*Σ(f(xi),1≤i≤n)
}
```

误差分析：f(x)抽样结果的方差$\sigma^2 =< f^2(x) > - < f(x) >^2$，积分估值与真实值之间的误差$En \approx \frac{\sigma}{\sqrt{n}} = \sigma m$（平均值的标准偏差）。

★5-3.Monte-Carlo积分法之三：重要抽样法(importance sampling method)

对于定积分$F = \int_a^b f(x)dx$，引入概率密度函数p(x)，它满足$\int_a^b p(x)dx = 1$，且p(x)在积分域内的函数形状与f(x)接近，即使得$\frac{f(x)}{p(x)}$的方差更小，这样定积分可改写为$F = \int_a^b [\frac{f(x)}{p(x)}] * p(x)dx$，它表示当x满足概率密度p(x)分布时，$\frac{f(x)}{p(x)}$的平均值$< \frac{f(x)}{p(x)} >$。 算法步骤： (1)根据概率密度函数p(x)产生随机点x，例如采用反变换法进行抽样。 (2)求出各抽样点x的函数值f(x)/p(x)，并将所有点的该函数值叠加起来除以抽样点数n就得到积分结果。

```
double ImpInt(double (*f)(double), double (*p)(double), double (*s)(), unsigned long n,
double &sigma){
//重要抽样法进行一维数值积分：f指向被积一元函数，p指向概率密度函数【p(x)必须满足在积分区间归一化】，s指向满足p(x)分布的抽样函数【它返回[a,b)区间按照p(x)分布的随机数】，n为抽样数，sigma记录单次抽样的标准偏差σ【平均值的标准偏差σm=σ/sqrt(n)】。
    double x, g, gaver=0.0, g2aver=0.0;
    unsigned long i;
    for(i=0;i<n;++i){
        x=(*s)();//根据概率密度函数p(x)抽样x
        g=(*f)(x)/(*p)(x);//计算f(x)/p(x)
        gaver+=g;
        g2aver+=g*g;
    }
    gaver/=n;
    g2aver/=n;
    sigma=sqrt(g2aver-gaver*gaver);//计算样本标准偏差σ
    return gaver;
}
```

应用举例：用重要抽样法计算定积分$F=\int_a^b f(x)dx$，其中$f(x)=e^{-x^2}$，取概率密度函数$p(x)=A*e^{-x}$。根据归一化条件，分析A的取值，写出相应的概率密度获取函数；采用反变换法推导x的抽样公式，写出相应的抽样函数。最后调用重要抽样算法估算积分值，并给出误差估算。

```
double a,b;//积分区间，定义为全局变量
double f(double x){ return exp(-x*x); }//被积函数
double p(double x){//概率密度函数，必须满足[a,b]区间归一化条件
    static double A=1.0/(exp(-a)-exp(-b));
    return A*exp(-x);
}
double cy(){//抽样函数，根据概率密度函数，采用反变换法得到x值
    static double B=exp(-a), A=1.0/(B-exp(-b));
    return -log(B-rnd()/A);
}
int main(){
    double Fn,s;
    unsigned int n;
    printf("Input a,b,n:");
    scanf("%lf,%lf,%ld",&a,&b,&n);
    Fn=ImpInt(f,p,cy,n,s);
    printf("Fn=%.10lg, s=%.10lg, sm=%.10lg\n",Fn,s,s/sqrt(n));
    printf("\nFinished!\n"); getch(); return 0;
}
```

★5-4.Metropolis抽样算法产生满足某概率密度函数p(x)（不一定归一化）分布的随机数的方法步骤。

抽样算法步骤：（1)选择一个尝试位置xt=x[i]+δi，δi是[-δ,δ)均匀分布随机数；（2)计算w=p(xt)/p(x[i])；（3)如果w>=1或rnd()<w，则接受改变x[i+1]=xt；否则不接受改变，即x[i+1]=x[i]。

```
void Metropolis(double (*p)(double),double &x,double delta,unsigned long &Na){
//Metropolis算法：产生{x}，使其满足概率密度函数p(x)分布，delta为随机行走最大步长，Na为接受改变的步数
    double xt=x+delta*2*rnd()-delta;
    double w=(*p)(xt)/(*p)(x);
    if(w>=1.0||rnd()<w){
        x=xt; ++Na;
    }
}
```

应用举例：用Metropolis抽样算法计算如下积分：$F=\frac{\int_{-\infty}^{+\infty} f(x)*p(x)dx}{\int_{-\infty}^{+\infty} p(x)dx}=<f(x)>$【x是满足概率密度函数p(x)分布的随机数，p(x)可以不必归一化】，其中$f(x)=x^2, p(x)=e^{-x^2/2}$

```
double f(double x){ return x*x; }
double p(double x){ return exp(-x*x/2); }
int main(){
    double x=0.0,faver=0.0,delta;
    unsigned long n,i,Na=0;
    printf("Input x0,delta,n:");
    scanf("%lf,%lf,%ld",&x,&delta,&n);
```

```
    for(i=1;i<=n;++i){
        Metropolis(p,x,delta,Na);
        faver+=f(x);
    }
    faver/=n;
    printf("Fn=%.14lg, Na=%lu\n",faver,Na);
    printf("\nFinished!\n"); getch(); return 0;
}
```

★6-1.二维随机行走的算法步骤

设行走者向上、右、下、左等4个方向行走的概率分别为p1、p2、p3、p4（p1+p2+p3+p4=1），产生[0, 1)均匀随机数r，如果∑(pj,0≤j≤i-1) ≤ r < ∑(pj,0≤j≤i)，则让行走者往第i个方向走。比如： p1+p2 ≤ r < p1+p2+p3 ，则往第3个方向行走，即向下走。

```
void RandWalkStep2(double ps[], long &x, long &y){
//二维随机行走一步，概率和数组ps[4]={p1,p1+p2,p1+p2+p3,p1+p2+p3+p4}，其中的p1、p2、p3、p4分别表示向上、右、下、左等4个方向行走的概率（p1+p2+p3+p4=1）。
//当产生的随机数ps[i-1]≤r<ps[i]，则发生第i个事件
    double r=rnd();
    int i;
    for(i=0;i<4;++i){ if(r<ps[i])break; }
    switch(i){
        case 0://向上走
            ++y; break;
        case 1://向右走
            ++x; break;
        case 2://向下走
            --y; break;
        case 3://向左走
            --x; break;
    }
}
```

★6-2.二维随机行走的统计规律

设行走者向上、右、下、左各方向行走的概率分别为pt、pr、pb、pl（pt+pr+pb+pl=1），每次行走的步长均为L，则行走N步后行走者的位置统计结果为： x平均值：$<x(N)>=N*(pr-pl)L$ x方差：
$<\Delta x^2(N)>=<x^2(N)>-<x(N)>^2=N*((pr+pl)-(pr-pl)^2)*L^2$ y平均值：
$<y(N)>=N*(pt-pb)L$ y方差：
$<\Delta y^2(N)>=<y^2(N)>-<y(N)>^2=N*((pt+pb)-(pt-pb)^2)*L^2$ 距离R方差：
$<\Delta R^2(N)>=<\Delta x^2(N)>+<\Delta y^2(N)>=N*(1-(pr-pl)^2-(pt-pb)^2)*L^2$