## 8.5 程序实例——变步长梯形积分算法

在实际工程应用中，许多定积分的求值都是通过计算机用数值计算的方法近似得到，原因是很多情况下，函数本身只有离散值，或函数的积分无法用初等函数表示。在这个例子中，我们介绍最基本的变步长梯形法求解函数的定积分的方法。

### 8.5.1 算法基本原理

我们只考虑最简单的情况，设被积函数是一个一元函数，定积分表达式为：

$$I=\int_a^b f(x)dx \tag{8-1}$$

积分表示的意义是一元函数 $f(x)$ 在区间 $a$ 到 $b$ 之间与 $x$ 轴所夹的面积，如图 8-1 所示。

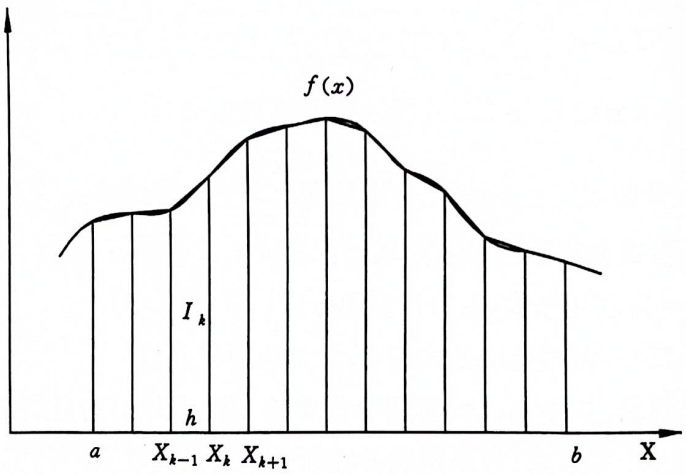


图 8-1 梯形积分原理示意图

如果上述积分可积，对于任意整数 $n$，取 $h=(b-a)/n$，

记 $$X_k=a+k*h$$

$$I_k=\int_{x_{k-1}}^{x_k} f(x)dx$$

则有： $$I=\sum_{k=1}^{n} I_k$$

在小区间 $[X_{k-1},X_k]$ 上，取 $I_k$ 的某种近似，就可以求得 $I$ 的近似值。

在区间 $[X_{k-1},X_k]$ 上取 $I_k=(f(X_{k-1})+f(X_k))*h/2$，就是梯形积分。梯形法积分的一个简单理解就是，把原积分区间划分为一系列小区间，在每个小区间上都用小的梯形面积来近似原函数的积分，当小区间足够小时，我们就可以得到原来积分的近似值。这时，整个积分结果可以近似表示为

$$T_n=\sum_{k=0}^{n-1}\frac{h}{2}[f(x_k)+f(x_{k+1})] \tag{8-2}$$

在使用这个积分公式之前必须给出合适的步长 $h$，步长太大精度难以保证，而步长太

小会导致计算量的增加。因此，实际计算往往采用变步长的方法，即在步长逐次减半(步长二分)的过程中，反复利用上述求积公式进行计算，直到所求得的积分结果满足所要求的精度为止。

把积分区间[$a$,$b$]分成$n$等分，就得到$n+1$个分点，按照公式(8-2)计算积分值$T_n$，需要计算$n+1$次函数值。如果积分区间再二分一次，则分点增加到$2n+1$个，我们把二分前后的积分值联系起来考察，分析它们的递推关系。

在二分之后，每一个积分子区间[$x_k$,$x_{k+1}$]经过二分后只增加了一个分点$x_{k+1/2}=(x_k+x_{k+1})/2$，二分之后这个区间的积分值为

$$\frac{h}{4}[(f(x_k)+2f(x_{k+\frac{1}{2}})+f(x_{k+1})] \tag{8-3}$$

注意，这里的$h=(b-a)/n$，还是二分之前的步长。把每个区间的积分值相加，就得到二分之后的积分结果

$$T_{2n}=\frac{h}{4}\sum_{k=0}^{n-1}[f(x_k)+f(x_{k+1})]+\frac{h}{2}\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}}) \tag{8-4}$$

利用二分前的积分结果(8-2)，就可以得到递推公式

$$T_{2n}=\frac{1}{2}T_n+\frac{h}{2}\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}}) \tag{8-5}$$

对于实际问题，我们采取的一般步骤为：

第一步，取$n=1$，利用公式(8-2)计算积分值；

第二步，进行二分，利用递推公式(8-5)计算新的积分值；

第三步，进行判断，如果两次计算积分值的差在给定的误差范围之内，二分后的积分值就是所需的结果，计算结束。如果两次计算积分结果之差在误差范围之外，返回第二步，继续运行。

### 8.5.2　程序设计分析

从上面的算法分析可以看到，我们面临的主要问题有两个，一是被积函数值的计算，每次计算积分值，都要在小的积分区间上计算函数的值；二是变步长梯形积分的实现。

程序中可定义两个抽象类：函数类 F 和积分类 Integ。它们所包含的纯虚函数为

```
virtual double operator ()(double x) const=0;          //纯虚函数重载运算符()
virtual double operator ()(double a,double b,double eps) const=0;
```

它们都重载了运算符()，前者是计算函数在 x 点处的值，后者是计算被积函数在区间[a,b]之间，误差小于 eps 的积分值，具体的函数实现由它们的派生类给出。这样做的好处是当派生类给出不同的函数实现时，通过这两个重载运算符，就可以计算不同被积函数的值或给出不同的求积分方法(本例中只给出变步长梯形法，还可以通过派生新类，加入其他积分算法)。

积分计算过程中，在梯形积分类 Trapz 的成员函数中需要访问函数类 Fun 的函数成

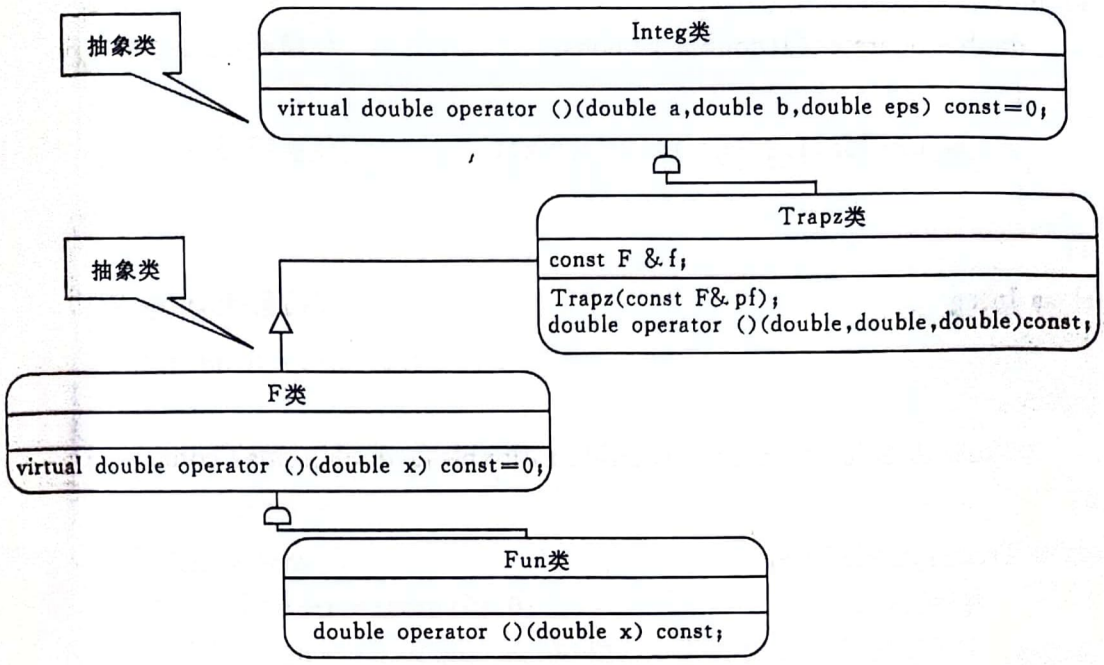


图 8-2　变步长梯形积分类的关系图

员，在 Trapz 类中加入了公有数据成员 f(F 类的引用)，利用虚函数的特性和赋值兼容规则，通过这个指针，我们可以访问到它正在指向的 F 类派生类对象的成员函数。

### 8.5.3　源程序及说明

**例 8-6　变步长梯形积分法**

整个程序分为三个独立的文档，Trapzint.h 文件包含类的声明，Trapzint.cpp 文件包含类的成员函数实现。文件 intmain.cpp 包含程序的主函数，主函数中定义了函数类 Fun 和梯形积分类 Trapz 的对象，通过这些对象求一个测试函数在某给定区间的积分(表达式 8-6)值，误差(eps)为 $10^{-7}$。

$$I=\int_1^2 \frac{\log(1+x)}{1+x^2}dx \tag{8-6}$$

```
//Begin of file Trapzint.h                          //文件一，类声明
#include <iostream.h>
#include <math.h>
class F                                             //抽象类 F 的声明
{
public:
    virtual double operator ()(double x) const=0;   //纯虚函数重载运算符()
};
class Fun:public F                                  //公有派生类 Fun 声明
{
```

```
public:
    double operator()(double x) const                    //虚函数的内联实现
    {
        return log(1.0+x)/(1.0+x*x);                     //被积函数
    }
};
class Integ                                              //抽象类 Integ 声明
{
public:
    virtual double operator ()(double a,double b,double eps) const=0;
};
class Trapz:public Integ                                 //公有派生类 Trapz 声明
{
public:
    Trapz(const F&pf):f(pf){}                            //构造函数
    double operator ()(double a, double b,double eps) const;
private:
    const F &f;                                          //私有成员,F 类对象的指针
};
                                                         //End of file Trapzint.h
```

经过公有派生,Fun 类获得了除构造函数、析构函数之外的 F 类的全部成员。由于基类 F 包含一个纯虚函数,是抽象类,因此在其派生类 Fun 中给出了基类继承来的纯虚函数成员的具体实现,即表达式 8-6 中的被积函数

$$f(x)=\frac{\log(1+x)}{1+x^2} \tag{8-7}$$

由于这个成员函数与基类函数的名称、参数、返回值完全相同,系统自动认定它是虚函数,不再需要显式声明。如果要求另外一个函数的积分,只要从基类 F 中重新派生一个类,比如 Fun1,把函数写为 Fun1 的纯虚函数的实现即可。

公有派生之后,梯形积分类 Trapz 继承了积分类 Integ 的成员。Integ 是抽象类,在派生类 Trapz 中给出基类 Integ 纯虚函数成员的具体实现,这个实现部分是在文件 Trapzint.cpp 中给出的。

```
//Begin of file Trapzint.cpp                             //文件二,类实现
#include "Trapzint.h"                                    //包含类的声明头文件
double Trapz::operator ()(double a,double b,double eps) const
{                                                        //积分运算过程,重载运算符()
    int done(0);                                         //是 Trapz 类的虚函数成员
    int n;
```

```
    double h,Tn,T2n;
    n=1;
    h=b-a;
    Tn=h*(f(a)+f(b))/2.0;                    //计算 n=1 时的积分值
    while(! done)
    {
        double temp(0.0);
        for(int k=0;k<n;k++)
        {
            double x=a+(k+0.5)*h;
            temp+=f(x);
        }
        T2n=(Tn+h*temp)/2.0;                 //变步长梯形法计算
        if(fabs(T2n-Tn)<eps) done=1;         //判断积分误差
        else                                 //进行下一步计算
        {
            Tn=T2n;
            n*=2;
            h/=2;
        }
    }
    return T2n;
}
//End of file Trapzint.cpp
```

上述函数以运算符()重载的方式实现求函数 f 在区间[a,b]上的积分结果，积分计算误差由 eps 来控制，函数 f 是抽象类 F 的引用，其具体实现由 F 的派生类 Fun 给出，返回值是积分结果。

```
//b8_6.cpp                                   //文件三，主函数
#include "Trapzint.h"                        //类声明头文件
#include "iomanip.h"
void main()                                  //主函数
{
    Fun f1;                                  //定义 Fun 类的对象
    Trapz trapz1(f1);                        //定义 Trapz 类的对象
    cout<<"TRAPZ Int:"<<setprecision(7)<<trapz1(0,2,1e-7)<<endl;
                                             //计算并输出积分结果
}
```

在程序的主函数部分，定义了 Fun 类的对象 f1 和梯形积分类 Trapz 的对象 trapz1，

并用 f1 来初始化对象 trapz1。这时，梯形积分类 Trapz 的数据成员 f(抽象类 F 的引用)就成为 Fun 类对象 f1 的别名。在对象 trapz1 中对基类 F 引用的操作，就是对 Fun 类对象 f1 进行的。通过对象 trapz1 调用重载的运算符()，就完成了求函数在区间[0,2]之间、误差小于 $10^{-7}$的积分的计算。

### 8.5.4　运行结果与分析

程序运行结果为：

```
TRAPZ Int:0.5548952
```

如果要求其他函数的积分，可在本程序的基础上修改。首先在 Fun 类的虚函数中给出被积函数的原形，然后在主函数输出语句的 trapz1 对象调用重载运算符()时给定积分区间 a 和 b 以及积分误差控制 eps 的值。如果要使用其他的积分算法，比如变步长辛普生(Simpson)算法等，就需要重新派生一个新类，比如名为 Simpson 的类，将该类中运算符()重载函数改写为辛普生算法，同时，在主函数中定义一个 Simpson 类的对象来计算积分值。

这是一个使用类来解决稍微复杂一些的实际问题的例子，涉及到了面向对象编程的几乎全部核心内容，类的继承、运算符重载、抽象类等，尤其是利用抽象类来实现多态部分，需要认真学习、理解，当然最好是自己上机调试、运行一遍。

## 8.6　程序实例——人员信息管理程序

在第 7 章中，我们以一个小型公司的人员信息管理程序为例，说明了类的派生过程及虚函数和虚基类的应用。但是例 7-10 的程序存在着两个不足：

① 基类的成员函数 pay()和 displayStatus()的函数体均为空，在实现部分仍要写出函数体，显得冗余。

② 在 main()函数中，建立了四个不同类的对象，对它们进行了类似的操作，但是却重复写了四遍类似的语句，程序不够简洁。

本节中，我们应用虚函数和抽象类对该程序进行改进，解决上述不足。

本例的类设计与第七章例 7-10 基本相同，只是在基类 employee 中将 pay()和 displayStatus()设计为纯虚函数。这样，在主函数中便可以依据赋值兼容规则用基类 employee类型的指针数组来处理不同派生类的对象，这是因为当用基类类型的指针调用虚函数时，系统会执行指针所指的对象的成员函数。

另外，由于不同类的对象调用升级函数 promote()需要给予不同的实参值(指定提升的级数)，所以难以利用循环语句在循环体中对各类对象统一处理。因此本例中将基类 employee 的成员函数 promote()定义为虚函数，各派生类中再定义同名函数，在派生类的 promote()函数中再以不同的实参调用基类 employee 的成员函数 promote()。类图如图 8-3 所示。