文章编号:1000 - 582X(2003)08 - 0130 - 05

面向对象的边界元程序设计*

袁政强[1], 肖 捷[1], 祝家麟[2]
(1. 重庆大学 土木工程学院,重庆 400045; 2. 重庆大学 数理学院,重庆 400044)

摘 要:按照面向对象的程序设计方法,遵循边界元分析的本质,建立了有关描述边界元模型的类,用链表方式实现结点、配置点、边界单元和内部单元的数据存放、用多态性实现单元的自由链接,方便的实现了单元增减、复连通区域和同一程序解不同问题等功能。采用 VC + + 编制了边界元配置法的数值计算程序,并给出了三维 Laplace 方程在球体区域上的算例。并将此方法的计算结果与精确解进行了比较,计算结果吻合良好。

关键词:面向对象;边界元;链表结构
中图分类号:TP311.132.4;o242.21 **文献标识码**:A

边界元方法是求解工程问题的数值方法,与常用的有限元法[1]相比有数据准备简单、降低计算一维数、计算速度快等优点,但它的理论复杂,编写的程序通用性差。传统的边界元程序设计一般采用结构化的程序设计方法和结构化语言(如 FORTRAN),其程序一般只能用于求解一个问题,程序的扩展能力有限,代码的重复利用率低,调试复杂。采用面向对象的程序设计,可以使程序具有封装性、继承性和多态性等优点,使得程序设计概念清楚,调试容易,代码的重复利用率高。面向对象的编程技术[2]已成为现代程序设计的主要方法之一。文献[3 - 4]中已给出了面向对象的有限元方法,还没有见到关于面向对象的边界元方法程序设计等的文章。边界元方法又分为边界配置法和边界有限元法,采用面向对象的程序设计,是将求解的问题进行分类和抽象。边界元是一种求解各种问题的方法,将边界元法作为抽象方法类,具体求解的问题是方法类的派生类。方法类中包含的各种问题都共有的数据,如:结点集合、单元集合、总体刚度矩阵和形成总体刚度矩阵的函数等。在方法类级的形成刚度矩阵函数中,包含通用的配置法形成刚度矩阵的计算过程,即对每一个配置点计算所有边界单元的单元刚度矩阵和由边界条件得到的右端项。单元集合是由边界单元和内部单元组成的单元链表,C + + 中的多态性可以让由一个抽象单元派生的各种单元放入一类链表中,抽象单元派生的边界单元和内部单元由抽象单元提供的形成刚度矩阵接口对不同单元调用各自的刚度矩阵计算函数进行计算,对于内部单元,刚度矩阵的计算只放入结果线性方程的右端向量中。这样形成刚度矩阵的计算由三级函数完成,方法类级的计算只将计算配置点作为参数调用单元级的刚度矩阵函数,单元级的刚度矩阵函数是将第三级的刚度矩阵函数得到的单元刚度矩阵和右端项放入总体刚度矩阵和线性方程组的右端项中,前两级的刚度矩阵函数对于所有的问题都不变,对具体的问题只增加具体问题的单元和相应的第三级刚度矩阵函数。第三级的刚度矩阵函数根据配置点和本单元的几何数据计算出基本解和基本解对边界的导数对应的 2 个子刚度矩阵,2 个子刚度矩阵由边界条件数和边界条件值形成单元刚度矩阵和单元右端项,单元刚度矩阵的行数与计算配置点自由度(Dof)相同,列数是单元配置点数(nd)乘以配置点自由度,即矩阵阶是 Dof × (nd * Dof)。当采用常单元时,单元刚度矩阵是方阵,阶为配置点自由度数 Dof × Dof。求解各种问题(如:Laplace 二维问题、Laplace 三维问题)的边界元配置法采用相同的方法类派生问题求解类,求解问题不同时只增加具体单元类和单元刚度矩阵计算。大大简化了程序的结构,方便的实现同

* 收稿日期:2003 - 04 - 15
基金项目:国家自然科学基金(19171197)
作者简介:袁政强(1962 -),男,湖南长沙人,重庆大学副研究员,博士生,主要从事工程力学和钢筋混凝土结构研究。

一程序对多类问题求解。文章还给出了三维 Laplace 问题常数单元的计算算例，采用 Visual C++，在 Windows98 上实现算例计算。结果表明，程序设计和调试周期较传统设计方法明显缩短，代码的利用率也明显提高。

1 程序设计

采用边界元配置法分析问题时，边界积分方程为：

$$\theta_{ki}u_i(x) = \int_{\Gamma_y}\frac{\partial u_i}{\partial n_x}(y)E^*_{ki}(y-x)d\Gamma_y - \int_{\Gamma_y}u_i(y)\frac{\partial E^*_{ki}}{\partial n_x}(y-x)\mathrm{d}\Gamma_y + \int_\Omega E^*_{ki}(y-x)f_i\mathrm{d}\Omega_y \quad (1)$$

其中：下标 i,k 代表求解问题的自由度(Dof)，对于 Laplace 问题，$i,k=1$；对于三维弹性力学问题，$i,k=1,2,3$。θ_{ki} 与 x 点边界 Γ 的光滑程度有关，如果 Γ 与 x 点处光滑，则 $\theta_{ki}(x)=\frac{1}{2}\delta_{ik}$。此式也用于计算内点处的函数值，此时取 $\theta_{ki}=I$，I 为单位矩阵。$E^*_{ki}(y-x)$，$\frac{\partial E^*_{ki}}{\partial n_x}(y-x)$ 基本解、基本解对边界方向导数值。对于不同问题的基本解可参看文献[5]。

边界单元的结点分为参与形成刚度矩阵的配置点和不参与形成刚度矩阵的几何结点。单元是线性插值单元时，单元的配置点与结点相同。方程为齐次函数时，内部单元的数据可以不参与计算。在形成单元刚度矩阵的计算中，单元对某一配置点 P 计算，得到的矩阵数据放入 P 配置点对应的行，单元配置点对应的列中。在一般的 FORTRAN 程序中，得到的单元刚度矩阵是 2 个边界积分得到的 2 个单元刚度矩阵。在程序中，将 2 个单元刚度矩阵经边界条件计算以后得到的是一个单元刚度矩阵和一个右端项向量。最终得到的总体刚度矩阵是满矩阵。边界元分析的主要数据包括：1）描述边界元分析的整体数据，如边界单元总数、内部单元总数、结点总数、问题的维数等；2）结点数据，包括结点坐标；3）配置点数据，包括坐标、自由度、位移、位移方向导数边界条件数等；4）边界单元数据，包括单元结点数目、配置点数目、单元类型、高斯积分点数据等；5）内部单元数据，包括单元结点数目、配置点数目、单元类型、高斯积分点数据等；因此，按照面向对象的程序设计方法，将原来的数据归并在相关的类中，主要有：1）结点数据类和相关的方法；2）配置点数据类和相关的方法；3）边界单元数据类和相关的方法；4）内部单元数据类和相关的方法。

2 面向对象的类

2.1 结点数据类

结点数据类用于边界元的配置点和几何计算点，如果条件数 BC 的值是 -1 时，表示结点是几何计算点。结点定义的定义：

```
class NodeData{
protected:
int Ind; //结点指示值(序号)
unsigned int NEquation; //结点在方程中的开始编号,
//程序自动计算
int NodeFree,Dim; //结点自由度数、空间维数
Vector<int> *BC; //用于指示结点的自由度的条件
//数,0:表示已知位移,1:表示已知面力
Vector<float> *Coor,*BCV; //结点坐标;边界条
//件值,0:表示已知位移,1:表示已知面力
Vector<float> *Value[5];//非线性分析和动力分析
//使用的存放的速度、加速度等值 public:
NodeData::NodeData(int iInd,int iNodeFree,int iDim);
//构造函数
~NodeData(){delete BC;delete Coor;}; //析构函数
public:
int GetInd(){return (Ind);}; //取得结点序号
NodeData *GetThis(int iInd);//根据序号返回本结点
//地址
Vector<int> *GetBC(){return BC;};
Vector<float> *GetBCV(){return BCV;};
Vector<float> *GetForce(){return Force;};
void SetForce(Vector<float> *F){Force=F;};
int GetDof(); //取得结点自由度数
    Vector<float> *GetCoor(){return Coor;};
};
```

值数组 Value[6] 根据求解的问题放入值的内容，坐标、力、位移值是确定的，温度、速度、加速度和历史数据，根据问题的需要申请。在不需要时，存放 NULL。

2.2 材料数据类

材料数据包括的内容，根据材料类型的不同而不同，放在 Mate 数组中的值可以使计算中需要的中间数据。

```
class Material{
protected:
int iInd; //材料指示数
Vector<float> *Mate; //存放杨氏模量、泊松比、材
```

```
//料密度、二维单元的厚度
public:
Material(int Ind); //输入材料数据
Material *GetThis(int Ind); //根据指示数取得材料
//指针
};
```

2.3 单元数据抽象类与 Laplace 单元类

抽象类所表达的概念广泛,不是一种具体的对象,它的唯一用处是为其它类提供基类,其它类可以从这里继承共有接口。"一个接口,多个算法"就是所谓的多态性。比如:边界单元的计算和体积单元的刚度矩阵计算是不同的。用抽象单元的单元刚度矩阵计算一个接口,可实现不同单元刚度矩阵的计算。只要调用的是单元刚度计算函数,不必特意选择,编译程序会动态的根据单元的类型自动选用相应的具体函数去计算,这种动态的由编译程序来决定调用函数的过程叫动态联编。动态联编是面向对象编程的关键,以前的许多过程化编程软件经改造后也能提供对象和指针,如:FORTRAN 90,但他们没有动态联编技术都不能称为面向对象的编程工具。

抽象单元的虚函数就是实例类单元的接口,虚函数是不用编写的。抽象单元类定义:

```
class Element{
protected:
int iType; //单元类型指示
Material *pMate;//指向材料类的指针(材料数据)
List<NodeData> *pNode; //指向结点类的链表(单
//元联络性数据)
public:
Element(int Type,Material *pM,int NodeNum);
List<NodeData> *GetNode();
void GetStiffMatrix(int &num,NodeData &no,Matrix<
float> &A,Vector<float> &F);
//虚函数 计算单元刚度矩阵并放入总刚度矩阵
virtual void StiffBEM(int &num,NodeData &no,Matrix<
float> &,int,Vector<float> &)=0;
virtual void GetStrain()=0; //虚函数 计算结点应变
virtual void GetStress()=0; //虚函数 计算结点应力
};
```

抽象单元的成员函数 GetStiffMatrix 是形成刚度矩阵的通用函数,对于不同空间维数和不同自由度问题都通用。在函数中调用了虚函数 StiffBEM,StiffBEM 师不同单元刚度矩阵计算的统一接口。函数如下:

```
void Element::GetStiffMatrix(int &num,NodeData &no,
Matrix<float> &A,Vector<float> &F){
    setIndex(Index,Index1); //求配置点和单元在总
//刚度矩阵中的位置编号
    StiffBEM(num1,no,a,0,f); //计算单元刚度矩阵
//将单元刚度矩阵放入总刚度矩阵之中
    if(num1==num){ // 边界单元积分
    if(num){
    for(i=1;i<=nd;i++)for(j=1;j<=Dof;j++) for(k=1;k<=Dof;k++) //刚度矩阵不对称
    A(Index1(j),Index((i-1)*Dof+k))+=(a)(i,(j-1)*Dof+k); //将单元刚度矩阵放入总刚度
//矩阵
    for(j=1;j<=Dof;j++)
    F(Index1(j))+=f(j);}
    else {// 当 num=0 时,a 是对角元素和
    for(i=1;i<=Dof;i++) for(j=1;j<=Dof;j++)
    if((*no.GetBC())(j)==1) // KODE=0 不
//交换 KODE=1 要交换
    A(Index1(i),Index1(j))=a(1,(i-1)*Dof+j);
    else
    F(Index1(j))-=a(1,(i-1)*Dof+j)*(*no.GetBCV())(j);};}
    else { // 内部单元计算
    for(i=0;i<nd;i++) for(j=1;j<=Dof;j++) for(k=1;k<=Dof;k++)
    if((*pn[i]->GetBC())(k)==-1) // 对于内部单元结点,条件数为-1
    F(Index1(j))-=a(i+1,(j-1)*Dof+k)*(*pn[i]->GetBCV())(k);};
};
```

Laplace 问题的内部单元和边界单元对是抽象单元类的派生,它们的定义:

```
class E_Laplace : public Element {//Laplace 边界单元
public:
E_Laplace(Material *pM):Element(pM){Dim=2;Dof
=1;nd=2;type=E_LAPLACE;};
E_Laplace(List<NodeData> &f,int *node):Element
(f,node);
void StiffBEM(int &num,NodeData &no,Matrix<float>
&,int,Vector<float> &);// 计算单元刚度矩阵和
//内力
};
```

```
class E_Laplacei : public Element {//Laplace 内部单元
public:
E_Laplacei(Material * pM);
E_Laplacei(List < NodeData > &f,int * node);
void StiffBEM(int &num,NodeData &no,Matrix < float >
&,int,Vector < float > &);// 计算单元刚度矩阵和
//内力
};
```

边界单元的 StiffBEM 函数得到的刚度矩阵要放入总刚度矩阵，而内部单元的 StiffBEM 函数计算结果只放入线性方程的右端项中。

2.4 边界元方法主类和问题派生类

边界元主类包含边界元分析的主要数据和实现方法。它的定义如下

```
class BemMethod{
private:
List < NodeData > * HeadNode; //结点链表
List < Element > * HeadElement; //单元链表
List < Material > * HeadMate; //材料链表
Vector < float > * F; // 求解方程时的右端向量
Matrix < float > * K; // 刚度矩阵
public:
BemMethod();
int Counter();// 计算结点各自由度的方程编号
int outputElement();// 输出单元数据
void GetStiffMatrixBEM();// 计算单元刚度
int solver();//方程求解
int BemMethod::GenGaussPoint();//计算高斯点的应
//变和应力并输出
};
```

面向对象的程序设计注重程序的重用技术[6]。给出的刚度矩阵函数是对任何空间维数和不同自由度都通用的函数。刚度矩阵函数如下：

```
void BemMethod::GetStiffMatrixBEM(){
    NodeData * pw; int i,num; K.ReSet();// 在计
//算刚度矩阵之前，将刚度矩阵 K 置零，
    ListIterator < NodeData > len( * HeadNode);ListIt-
erator < Element > lei( * HeadElement);
    while(len.NextNotNull()) { //对配置结点循环
    pw = len.Next();i = 1;lei.First();
    while(lei.NextNotNull()) { //对边界单元和内部
//单元循环
    num = i + +;lei.Next() - > GetStiffMatrix(num,
* pw,K, * F);};
    lei.First();num = 0;
    lei.Next() - > GetStiffMatrix(num, * pw,K, *
F); // 将对角线元素放入刚度矩阵中
    };
};
```

边界元方法类是一个基本方法类，对于具体求解问题的类可由边界元方法类派生。Laplace 分析方法类：

```
class F_Laplace: public BemMethod{
public:
    F_Laplace(int dim,int free); //构造函数，生成
Laplace 边界元方法类
    int Anlysis();// 边界元分析程序
};
int F_Laplace::Anlysis(){
    Counter(); //计算方程编号和申请总刚度矩阵
    GetStiffMatrixBEM(); // 形成总刚度矩阵
    solver(); // 方程求解
    outputDisplace(); // 计算内部函数值及输出
};
```

面向对象的 C + +语言与结构化的语言最大的不同在于它的注意点是数据而不是函数。数据定义的好坏确定了整个软件的处理方法和软件处理问题的能力。下面对以上定义的数据结构作必要的说明，从说明中可以看出，用传统 FORTRAN 语言无法解决的问题，用 C + +很方便的得到解决。用传统 FORTRAN 很难解决的问题，用 C + +解决起来就变得容易。

3 类的实例化

算例采用的区域为半径为单位长度的球体，区域边界球面剖分为等边三角形组成的面。数值积分解采用三角形高斯 7 点积分方法计算积分（结果见表 1）。

```
#include"BEM.h"
void main(){
    F_Laplace * b;
    b = new F_Laplace(3,1);
    b - > Anlysis();//
};
```

表 1 Laplace 方程解为：x^*x-y^*y 时的计算结果

内点	坐标 X	坐标 Y	坐标 Z	单元数	本算法解	单元数	本算法解	精确解
P_1	0.2	0.0	0.0	128	0.038 736 6	512	.039 704 6	.040 000
P_2	0.0	0.3	0.0	128	-0.087 234 8	512	-0.089 348 9	-.090 000
P_3	0.0	0.0	0.4	128	-1.996 38e-17	512	-8.085 39e-20	.000 000
P_4	0.5	0.0	0.0	128	0.243 291	512	0.248 360	.250 000
P_5	0.0	0.6	0.0	128	-0.351 460	512	-0.357 842	-.360 000
P_6	0.0	0.0	0.7	128	-3.834 91e-17	512	-6.508 48e-17	.000 000

4 结 语

以上给出了边界元分析的基本结构类，在边界元分析中还牵连许多问题，如求解线性方程组的方法，这些都在矩阵模板类和向量模板类中定义。矩阵模板类和向量模板类[6]中给出了矩阵与矩阵运算、向量与向量运算和矩阵与向量的混合运算。因这些大多是计算方法的内容，我们将在另文中给出。在文中给出的边界元方法类是最基本的方法类，具体问题（如：温度场分析、流场分析和污染物扩散分析）类可在基本方法类下继承，这使得加入具体问题的求解变得容易。

参考文献：

[1] ZIIENKIEWICZ O C. The Finite Element Method[M]. UK:McGraw-Hill Book Company(UK) Ltd,1977.

[2] HEKMATPOUR S. C 程序员的 C++指南[M]. 阎尤泽译. 北京：航空航天大学出版社，1992.

[3] FORD B W R, FOSCHI R O, STIENER S F. Object-oriented finite element analysis[J]. Computers and Structures, 1990,35:355-374.

[4] SCHOLZ S P. Element of an object-oriented FEM program in C++[J]. Computers and Structures, 1992,43:517-529.

[5] 祝家麟. 椭圆边值问题的边界元分析[M]. 北京：科学出版社，1991.

[6] 程纬，曹定华，张诚坚. 通用矩阵与矢量模板类的分析与应用[J]. 湖南大学学报，1999,26(5):97-101.

Object-oriented Boundary Element Method Programming

YUAN Zheng-qiang[1], *XIAO Jie*[1], *ZHU Jia-lin*[2]

(1. College of Civil Engineering, Chongqing University, Chongqing 400044, China;
2. College of Mathematics and Sciences, Chongqing University, Chongqing 400045, China)

Abstract: The object-oriented programming concept is applied to boundary element method. According to the nature of the boundary element analysis, the classes and their methods, which describe virtual element, node, material etc., have been developed and implemented using the object-oriented programming language C++. The elements, nodes and materials are stored by chain. Many types elements are stored mixedly by polymorphism characteristic. The instance is given to show the programming of boundary element method of Laplace equation of three dimensions on the sphere. The both this result and exacted result are inosculate.

Key words: object-oriented; boundary element method; chain

（编辑 姚 飞）