OpenCascade Matrix

eryar@163.com

摘要 Abstract：本文对矩阵作简要介绍，并结合代码说明 OpenCascade 矩阵计算类的使用方法。

关键字 Key Words：OpenCascade、Matrix、C++

一、引言 Introduction

矩阵的研究历史悠久，拉丁方阵和幻方在史前年代已有人研究。作为解决线性方程的工具，矩阵也有不短的历史。1693 年，微积分的发现者之一莱布尼茨建立了行列式论（theory of determinants）。1750 年，克拉默又定下了克拉默法则。1800 年代，高斯和威廉若尔当建立了高斯－若尔当消去法。

从其发展历史可以看出，从行列式论的提出到克拉默法则总共历时 50 多年之久，而在我们《线性代数》的课本中，也就是一章的内容，学习时间可能只有一个星期。难怪这些概念这么抽象晦涩，也是情有可原的，哈哈。不过，当理解这些概念，也就掌握了一个强大的工具，因为它也使有些计算统一、简单。

矩阵是高等代数中的常见工具，也常见于统计分析等应用数学学科中。在物理学中，矩阵用于电路学、力学、光学和量子物理中都有应用；计算机科学中，三维变换就是使用了矩阵的线性变换。矩阵的运算是数值分析领域的重要问题。

在计算机图形相关的软件中，矩阵是必不可少的重要工具。如：OpenGL、DirectX 中的变换都是用矩阵实现的；OpenCascade 中就有矩阵计算类（math_Matrix）；OpenSceneGraph 中也有矩阵的计算类（osg::Matrix）等等。矩阵在计算中主要用来表达线性变换，统一三维空间中物体的位置和方向变换。

本文主要介绍 OpenCascade 中矩阵类的使用方法。

二、矩阵的运算 Matrix Calculation

矩阵的运算主要有矩阵的加法、数与矩阵相乘、矩阵与矩阵相乘、矩阵的转置、方阵的行列式、共轭矩阵、逆矩阵。还有通过矩阵来求解线性方程组的解问题。

1. 矩阵的加法 Matrix Addition

定义：设有两个$m\times n$矩阵$A=(a_{ij})$和$B=(b_{ij})$，那么矩阵A与B的和记作$A+B$，规定为：

$$A+B=\begin{Bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{Bmatrix}$$

应该注意，只有当两个矩阵是同型矩阵时，才能进行加法运算。矩阵加法满足下列运算规律：设 A、B、C 都是 m×n 矩阵：

$$(i) \quad A+B=B+A$$
$$(ii) \quad (A+B)+C=A+(B+C)$$

2. 数与矩阵相乘 Matrix Multiplication

定义：数λ与矩阵A的乘积记作λA或$A\lambda$，规定为：

$$\lambda A=A\lambda=\begin{Bmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & \vdots & & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn} \end{Bmatrix}$$

数乘矩阵满足下列运算规律（设 A、B 为 m×n 矩阵，λ、μ 为数）：

$$(i) \quad (\lambda\mu)A=\lambda(\mu A)$$
$$(ii) \quad (\lambda+\mu)A=\lambda A+\mu A$$
$$(iii) \quad \lambda(A+B)=\lambda A+\lambda B$$

矩阵相加与矩阵相乘合起来，统称为矩阵的线性运算。

3. 矩阵与矩阵相乘 Matrix Multiplication

定义：设$A=(a_{ij})$是一个$m\times s$矩阵，$B=(b_{ij})$是一个$s\times n$矩阵，那么规定矩阵A与矩阵B的乘积是一个$m\times n$矩阵$C=(c_{ij})$，其中：

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{is}b_{sj}=\sum_{k=1}^{s}a_{ik}b_{kj}$$

$$(i=1,2,\cdots,m;\ j=1,2,\cdots,n)$$

并把此乘积记作：$\boldsymbol{C=AB}$

按此定义，一个 1×s 行矩阵与一个 s×1 列矩阵的乘积是一个 1 阶方阵，也就是一个数：

$$(a_{i1},a_{i2},\cdots a_{is})\begin{pmatrix}b_{1j}\\b_{2j}\\\vdots\\b_{sj}\end{pmatrix}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{is}b_{sj}=\sum_{k=1}^{s}a_{ik}b_{kj}=c_{ij}$$

由此表明乘积矩阵 AB＝C 的（i,j）元 c_{ij} 就是 A 的第 i 行与 B 的第 j 列的乘积。必须注意：只有当每一个矩阵（左矩阵）的列数等于第二个矩阵（右矩阵）的行数时，两个矩阵才能相乘。

矩阵的乘法虽不满足交换律，但仍满足下列结合律和分配律（假设运算都是可行的）：

$$(i)\qquad (AB)C=A(BC)$$

$$(ii)\qquad \lambda(AB)=(\lambda A)B=A(\lambda B)$$

$$(iii)\qquad A(B+C)=AB+AC\qquad (B+C)A=BA+CA$$

4. 矩阵的转置 Matrix Transpose

定义：把矩阵A的行换成同序数的列得到一个新矩阵，叫做A的转置矩阵，记作A^T。

$$A=\begin{pmatrix}1 & 2 & 0\\ 3 & -1 & 1\end{pmatrix}\xrightarrow{\text{转置}}A^T=\begin{pmatrix}1 & 3\\ 2 & -1\\ 0 & 1\end{pmatrix}$$

矩阵转置也是一种运算，满足下列运算规律（假设运算都是可行的）：

$$(i)\quad (A^T)^T=A$$

$$(ii)\quad (A+B)^T=A^T+B^T$$

$$(iii)\quad (\lambda A)^T=\lambda A^T$$

$$(iv)\quad (AB)^T=B^TA^T$$

5. 方阵的行列式 Determinant of the Square Matrix

定义：由n阶方阵A的元素所构成的行列式（各元素的位置不变），称为方阵A的行列式，记作$|A|$或$\det A$.

应该注意，方阵与行列式是两个不同的概念，n 阶方阵是 n×n 个数按一定方式排列成的数表，而 n 阶行列式是这些数（也就是数表 A）按一定的运算法则所确定的一个数。

由 A 确定｜A｜的这个运算满足下述运算规律（设 A、B 为 n 阶方阵，λ 为数）：

$$(i) \quad |A^T| = |A|$$

$$(ii) \quad |\lambda A| = \lambda^n |A|$$

$$(iii) \quad |AB| = |A||B|$$

6. 逆矩阵 Inverse of Matrix

定义：对于 n 阶矩阵 $\boldsymbol{A}$，如果一个 n 阶矩阵 $\boldsymbol{B}$，使 $\boldsymbol{AB}=\boldsymbol{BA}=\boldsymbol{E}$，则说矩阵 $\boldsymbol{A}$ 是可逆的，并把矩阵 $\boldsymbol{B}$ 称为 $\boldsymbol{A}$ 的逆矩阵，简称逆阵。

方阵的逆矩阵满足下列运算规律：

(i) 若A可逆，则A^{-1}亦可逆，且$(A^{-1})^{-1}=A$

(ii) 若A可逆，数$\lambda\neq 0$，则λA可逆，且$(\lambda A)^{-1}=\frac{1}{\lambda}A^{-1}$

(iii) 若A、B为同阶矩阵且均可逆，则AB亦可逆，且$(AB)^{-1}=B^{-1}A^{-1}$

7. 线性方程组的数值解法

在自然科学和工程中有很多问题的解决都归结为求解线性方程组或者非线性方程组的数学问题。例如，电学中的网络问题，用最小二乘法求实验数据的网线拟合问题，三次样条的插值问题，用有限元素法计算结构力学中的一些问题或用差分法解椭圆型偏微分方程的边值问题等等。

求解线性方程组的直接法主要有选主元高斯消去法、平方根法、追赶法等。如果计算过程中没有舍入误差，则此种方法经过有限步算术运算可求得方程组的精确解，但实际计算中由于舍入误差的存在和影响，这种方法也只能求得方程组的近似解。目前这种方法是计算机上解低阶稠密矩阵的有效方法。

高斯消去法是一个古老的求解线性方程组的方法，但由于它改进得到的选主元的高斯消去法则是目前计算机上常用的解低阶稠密矩阵方程组的有效方法。

用高斯消去法解方程组的基本思想是用矩阵行的初等变换将系数矩阵化为具有简单形式的矩阵，即三角形矩阵，再通过回代过程，求出方程组的解。

三、OpenCascade 矩阵计算

在 OpenCascade 中，矩阵的计算类是 math_Matrix，它可以有任意的行数和列数，实现了矩阵大部分的运算。以下通过程序实例来说明 math_Matrix 的使用方法：

```
/*
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*        File    : Main.cpp
*        Author  : eryar@163.com
*        Date    : 2013-07-17 21:46
*        Version : 1.0v
*
*    Description : Demonstrate how to use math_Matrix class.
*                  题目来自《线性代数》同济 第四版
*
*/

#include <math_Matrix.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")

int main(int argc, char* argv[])
{
    math_Matrix A(1, 3, 1, 3);
    math_Matrix B(1, 3, 1, 3);
    math_Matrix C(1, 3, 1, 3);

    // 1. Test Matrix Transpose and multiply.
    // 题目来自P53习题二 第3题
    // Set value of Matrix A
    A.Init(1);
    A(2, 3) = -1;
    A(3, 2) = -1;

    // Set value of Matrix B
    B(1, 1) = 1;    B(1, 2) = 2;    B(1, 3) = 3;
    B(2, 1) = -1;   B(2, 2) = -2;   B(2, 3) = 4;
    B(3, 1) = 0;    B(3, 2) = 5;    B(3, 3) = 1;

    C.TMultiply(A, B);

    cout<<"Matrix A: "<<endl;
    cout<<A<<endl;
    cout<<"Matrix B: "<<endl;
```

```
    cout<<B<<endl;
    cout<<"Matrix C: "<<endl;
    cout<<C<<endl;

    // 2. Test the Determinant of the Matrix.
    // 题目来自P12 例7
    math_Matrix D(1, 4, 1, 4);
    D(1, 1) = 3;    D(1, 2) = 1;    D(1, 3) = -1;    D(1, 4) = 2;
    D(2, 1) = -5;   D(2, 2) = 1;    D(2, 3) = 3;     D(2, 4) = -4;
    D(3, 1) = 2;    D(3, 2) = 0;    D(3, 3) = 1;     D(3, 4) = -1;
    D(4, 1) = 1;    D(4, 2) = -5;   D(4, 3) = 3;     D(4, 4) = -3;

    cout<<"Matrix D: "<<endl;
    cout<<D<<endl;
    cout<<"Determinant of Matrix D: "<<D.Determinant()<<endl;

    // 3. Calculate Inverse of the Matrix.
    // 题目来自P54 11题(3) 和P56 30题(1)
    math_Matrix E(1, 3, 1, 3);

    E(1, 1) = 1;    E(1, 2) = 2;    E(1, 3) = -1;
    E(2, 1) = 3;    E(2, 2) = 4;    E(2, 3) = -2;
    E(3, 1) = 5;    E(3, 2) = -4;   E(3, 3) = 1;

    cout<<"Inverse of the Matrix E: "<<endl;
    cout<<E.Inverse()<<endl;

    // P56 30题(1)
    math_Matrix F(1, 4, 1, 4);

    F(1, 1) = 5;    F(1, 2) = 2;
    F(2, 1) = 2;    F(2, 2) = 1;
    F(3, 3) = 8;    F(3, 4) = 3;
    F(4, 3) = 5;    F(4, 4) = 2;

    cout<<"Inverse of the Matrix F: "<<endl;
    cout<<F.Inverse()<<endl;

    return 0;
}
```

计算结果为：

```
Matrix A:
math_Matrix of RowNumber = 3 and ColNumber = 3
```

```
math_Matrix ( 1, 1 ) = 1
math_Matrix ( 1, 2 ) = 1
math_Matrix ( 1, 3 ) = 1
math_Matrix ( 2, 1 ) = 1
math_Matrix ( 2, 2 ) = 1
math_Matrix ( 2, 3 ) = -1
math_Matrix ( 3, 1 ) = 1
math_Matrix ( 3, 2 ) = -1
math_Matrix ( 3, 3 ) = 1

Matrix B:
math_Matrix of RowNumber = 3 and ColNumber = 3
math_Matrix ( 1, 1 ) = 1
math_Matrix ( 1, 2 ) = 2
math_Matrix ( 1, 3 ) = 3
math_Matrix ( 2, 1 ) = -1
math_Matrix ( 2, 2 ) = -2
math_Matrix ( 2, 3 ) = 4
math_Matrix ( 3, 1 ) = 0
math_Matrix ( 3, 2 ) = 5
math_Matrix ( 3, 3 ) = 1

Matrix C:
math_Matrix of RowNumber = 3 and ColNumber = 3
math_Matrix ( 1, 1 ) = 0
math_Matrix ( 1, 2 ) = 5
math_Matrix ( 1, 3 ) = 8
math_Matrix ( 2, 1 ) = 0
math_Matrix ( 2, 2 ) = -5
math_Matrix ( 2, 3 ) = 6
math_Matrix ( 3, 1 ) = 2
math_Matrix ( 3, 2 ) = 9
math_Matrix ( 3, 3 ) = 0

Matrix D:
math_Matrix of RowNumber = 4 and ColNumber = 4
math_Matrix ( 1, 1 ) = 3
math_Matrix ( 1, 2 ) = 1
math_Matrix ( 1, 3 ) = -1
math_Matrix ( 1, 4 ) = 2
math_Matrix ( 2, 1 ) = -5
math_Matrix ( 2, 2 ) = 1
math_Matrix ( 2, 3 ) = 3
math_Matrix ( 2, 4 ) = -4
```

```
math_Matrix ( 3, 1 ) = 2
math_Matrix ( 3, 2 ) = 0
math_Matrix ( 3, 3 ) = 1
math_Matrix ( 3, 4 ) = -1
math_Matrix ( 4, 1 ) = 1
math_Matrix ( 4, 2 ) = -5
math_Matrix ( 4, 3 ) = 3
math_Matrix ( 4, 4 ) = -3

Determinant of Matrix D: 40
Inverse of the Matrix E:
math_Matrix of RowNumber = 3 and ColNumber = 3
math_Matrix ( 1, 1 ) = -2
math_Matrix ( 1, 2 ) = 1
math_Matrix ( 1, 3 ) = -8.88178e-017
math_Matrix ( 2, 1 ) = -6.5
math_Matrix ( 2, 2 ) = 3
math_Matrix ( 2, 3 ) = -0.5
math_Matrix ( 3, 1 ) = -16
math_Matrix ( 3, 2 ) = 7
math_Matrix ( 3, 3 ) = -1

Inverse of the Matrix F:
math_Matrix of RowNumber = 4 and ColNumber = 4
math_Matrix ( 1, 1 ) = 1
math_Matrix ( 1, 2 ) = -2
math_Matrix ( 1, 3 ) = 0
math_Matrix ( 1, 4 ) = 0
math_Matrix ( 2, 1 ) = -2
math_Matrix ( 2, 2 ) = 5
math_Matrix ( 2, 3 ) = -0
math_Matrix ( 2, 4 ) = -0
math_Matrix ( 3, 1 ) = 0
math_Matrix ( 3, 2 ) = 0
math_Matrix ( 3, 3 ) = 2
math_Matrix ( 3, 4 ) = -3
math_Matrix ( 4, 1 ) = -0
math_Matrix ( 4, 2 ) = -0
math_Matrix ( 4, 3 ) = -5
math_Matrix ( 4, 4 ) = 8
```

线性方程组的计算方法请参考另一篇 blog：

使用 OpenCASCADE 的 Math 功能解线性方程组

http://www.cppblog.com/eryar/archive/2012/06/21/179629.html

四、结论 Conclusion

OpenCascade 的 math_Matrix 实现了矩阵的大部分运算，也可以作为一个小型的 C++矩阵计算库来使用，在没有安装 Matlab、Mathematica 等软件的情况下，可以使用这个库进行一些计算。

另外，对《计算方法》中算法感兴趣的读者可以参考 OpenCascade 其中代码的实现，加深对相关算法的理解。

五、参考文献 Bibliography

1. 同济大学应用数学系 线性代数 高等教育出版社 2003
2. 易大义，沈云宝，李有法 计算方法 浙江大学出版社 2002