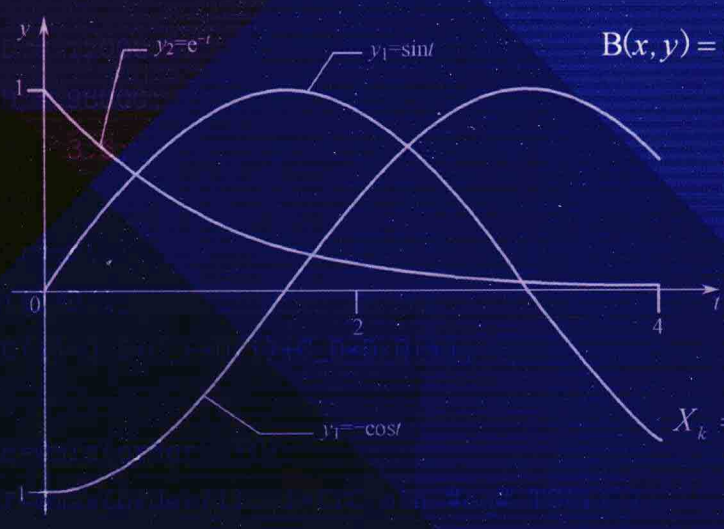
陈必红 著

用C++语言编写数学常用算法（修订版）



$$B(x,y)=B(y,x)=\int_0^1 t^{x-1}t^{y-1}dt,(x,y>0)$$

$$z=\sum_{i=k}^{k+7} y_i \prod_{\substack{j=k \\ j\neq i}}^{k+7}[(t-x_j)/(x_i-x_j)]$$

$$\Phi(x)=\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}}dt$$

$$X_k=\sum_{i=0}^{n-1} x_i e^{-2\pi j\frac{ki}{n}} \quad (k=0,1,\cdots,n-1)$$

$$\int_{-1}^{1}\varphi(t)dt\cong\sum_{k=0}^{n-1}\lambda_k\varphi(t_k)$$

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

用 C++语言编写数学常用算法

（修订版）

陈必红 著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书主要介绍用C++语言编写各种与实数和复数有关的常用数学算法的程序，包括线性代数、矩阵运算、实数方程求解、插值、拟合、数值积分、微分方程求解、特殊函数、函数变换、回归分析等。本书给读者提供两个方便实现数学算法的类，即矩阵类和函数类。书中所有程序均调试通过，并存放在电子资料包中。本书提供的类库为作者的独创，具有编程容易、效率高的特点。此外，本修订版增加了一章，专门介绍VC++编程，并给出了独特的子窗口技术。

本书可供科研人员、工程技术人员和程序员阅读使用，也可作为中、高等院校学生学习、研究及软件开发的参考书。

图书在版编目（CIP）数据

用C++语言编写数学常用算法/陈必红著. —修订本. 北京：电子工业出版社，2009.8
ISBN 978-7-121-09427-9

Ⅰ.用… Ⅱ.陈… Ⅲ. C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2009）第149069号

责任编辑：周 琰
印 刷：北京天宇星印刷厂
装 订：三河市皇庄路通装订厂
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编 100036
开 本：787×1 092 1/16 印张：18.75 字数：492千字
印 次：2009年8月第1次印刷
定 价：39.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

从事科学研究的人经常需要编写一些数学常用算法的程序，如矩阵求逆、解线性方程组、求解微分方程、作线性回归等。当然可以用 MATLAB 这样的软件工具来进行各种数学演算，但这样无法编写自己的软件产品、自己的程序，如数控机床的控制程序或自动驾驶仪程序就需要拿出自己的软件产品。大家都知道 C++是一种好的编程语言，可是往往搞科研的人没有更多的时间深入学习 C++语言，只能初通 C 语言的语法之后就上阵编程。

在编程过程中需要解一个数学问题时，读者通常是参考一些 C 语言实现数学算法的参考书。而现有的用 C 语言或 C++语言实现数学算法的参考书中，要么没有考虑到应用上的困难，例如，用 C 语言求逆矩阵的范例，如果矩阵特别大，内存放不下，就需要使用磁盘作缓存数组，可这样一来就不能直接使用范例程序，而不得不对范例程序进行适当修改，以适应具体需要，但改完之后再行调试，程序的可靠性就大受影响；要么就是过于偏重深奥的类编程知识，并搞出使用起来极为复杂的类库，使没有时间深入学习、熟练掌握 C++面向对象编程知识的读者很难读懂，而且这些类库也过分依赖于 Borland 公司提供的容器类库，虽然 Borland 公司提供了此类库的全部源程序，但这些程序的可读性很差，因此一旦需要修改，也非常困难。

本书是专门针对由于各种原因无法深入学习 C++知识，又希望很快编写出数学常用算法程序的技术人员编写的。本书给读者提供了一个可用的数学软件工具，能够节省读者编写数学计算程序的时间。阅读本书甚至不需要特别懂 C++面向对象的编程知识，只要初通 C++语言即可。

本书主要内容

第 1 章介绍如何使用本书提供的程序进行各种基本数学运算的编程，包括矩阵求逆、解线性方程组、求积分、求卡尔曼滤波、求傅里叶变换等各种算法，并给出了示例程序（本书的电子资料包中没有这些示例程序，因为它们实在太简单，敲几个字符就出来了）。该章还对各种算法的原理进行了简单介绍。

如果希望使用得更好一些，如对特别大的矩阵进行操作，或者进行好的出错处理（当接收到的数据不符合要求时，程序不是结束运行，而是通知用户出现错误，并让程序继续下去），或者想知道一个数组最多能够容纳多少个实数或复数，或者希望知道在几种可选的办法中哪一种效率比较高，这就需要阅读第 2 章。该章介绍了本书编写的类的更多信息，包括对参数进行控制、类变量的情况（如占多少内存、数据存放在哪里、怎样的编程效率比较高）等，以帮助读者掌握这些类的性能指标，并更好地使用这些类。看完该章，读者基本上已经能够利用本书编的程序编写合格的软件产品了。

如果还想对本书编写的类库进行更深入的了解，进行扩充，增加一些功能，做一些简单的修改，如为提高计算精度，用长双精度实数来进行所有的数学运算；或者将一些新的数学算法扩充到本书提供的类当中，那就需要阅读第 3 章。该章详细介绍了本书在设计数学计算时提供

的两大重要类，即矩阵类和函数类的基本思路。

第 4 章重点介绍了各个类库的作用，以及各类、各数据成员、各成员函数与全局函数的定义和基本结构。在本书电子资料包提供的源程序中包含有详细的中文注释，结合这些注释，读者可以掌握本书编写的数学计算类库的全部技术资料。

第 5 章针对微软的 VC++编译系统，用最少的篇幅介绍了 VC++的控制台程序和视窗程序的编写方法，尤其介绍了子窗口技术（子窗口技术是作者独创的，目前出版的 VC++编程指导书中基本上都没有提过子窗口技术）。为了说明子窗口技术，还给出了两个在科研中很有用的子窗口类（绘制函数曲线类和录制声音类）。对于 VC++的编程，还给出了示例程序，这些示例程序都在本书电子资料包的 examples 文件夹中。

本书电子资料包的使用

本书电子资料包包括书中全部 C++程序的源代码（一些小的示例程序除外），并用中文加以详细注释。读者可以到电子工业出版社的网站（网址为 http://www.phei.com.cn/download/09427.zip）或作者主办的应用数学家园网站（网址为 http://appmath.cn 或 http://应用数学.cn）下载。

电子资料包共有 BCMATH，VCMATH，CURVE，SOUND 和 examples 五个文件夹。前 4 章介绍的与数学常用算法有关的 C++类与函数的源代码是在 BCMATH 和 VCMATH 两个文件夹中，具体使用哪个文件夹，根据读者的编译系统而定。如果读者使用的是 Borland C++编译系统，则使用 BCMATH 文件夹中的所有文件（Borland C++的语法规则相当标准，因此该文件夹中的源代码也可以用在其他标准的 C++编译系统中）；如果读者使用的是微软的 Visual C++编译系统，则使用 VCMATH 文件夹中的所有文件（第 5 章详细介绍了 Visual C++编程）。文件夹 CURVE 给出了一个实用的绘制函数曲线的子窗口的源代码，文件夹 SOUND 给出了一些处理声音的 C++函数和子窗口的源代码，文件夹 examples 给出了一些 VC++编程的示例。

对于前 4 章的内容，读者根据使用的编译系统，将文件夹 BCMATH 或 VCMATH 的内容复制到存放源程序的目录下，并将其中的文件（包括后缀为.cpp 的 C++源程序文件及后缀为.h 的头文件）都加到读者创建的工程中即可。

其实电子资料包中只有以下几个文件：buffer.cpp, cbuffer.cpp, matrix.cpp, cmatrix.cpp, func.cpp, func1.cpp, cfunc.cpp, vfunc.cpp,buffer.h, cbuffer.h, matrix.h, cmatrix.h, func.h, func1.h, cfunc.h, vfunc.h，至于怎样把几个 C++程序连接到一起形成一个工程，请参考不同编译系统提供的有关技术指导书。

读者在阅读本书或使用本书电子资料包的程序时，如有任何疑问，可发电子邮件到 webmaster@appmath.cn 与作者联系。也可到作者主办的应用数学家园网站（网址为 http://appmath.cn 或 http://应用数学.cn）去查看疑难解答、错误更正等信息。

在本书编写过程中，得到了来自各方朋友的支持和帮助，特在此表示深切的谢意。

陈必红　博士

深圳大学数学与计算科学学院

目　录

第1章 矩阵类与函数类的基本用法

1.1 矩阵类的基本用法

在数学运算中最经常使用的是矩阵和向量，而向量其实也是多行一列的矩阵。因此，我们定义两个矩阵类：实数矩阵类 matrix 和复数矩阵类 cmatrix，以完成有关的数学运算。利用矩阵类可以很方便地进行矩阵的加、减、乘、求逆、求行列式；解线性方程组、求特征值和特征向量等运算。本节介绍实数矩阵类的用法。实数矩阵类在文件 matrix.h 中定义，所有算法的实现程序都在源文件 matrix.cpp 中。而复数矩阵类在文件 cmatrix.h 中定义，其算法的实现在源文件 cmatrix.cpp 中。而它们都受到缓存器类的支持，因此实数矩阵类的使用需要包括描述实缓存器的文件 buffer.h 和 buffer.cpp，而复数矩阵类的使用则需要包括描述复缓存器的文件 cbuffer.h 和 cbuffer.cpp。

本节仅介绍实数矩阵类 matrix 的用法。matrix 类的设计构思参见第 3.3 节，其详细的技术说明（包括 C++语句的定义代码）参见第 4.3 节，全部的源程序都在电子资料包中，且已经加进详细的中文注释。

1.1.1 举例说明

先通过一个例子来说明实数矩阵类 matrix 的用法。

假设有三个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 和 $\boldsymbol{C}$，$\boldsymbol{A}=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$，$\boldsymbol{B}=\begin{pmatrix}2 & 2\\ 4 & 6\end{pmatrix}$，$\boldsymbol{C}=\begin{pmatrix}3 & 3\\ 3 & 3\end{pmatrix}$，我们要计算 $2\boldsymbol{A}^{-1}\boldsymbol{B}+\boldsymbol{C}$，并将结果输出到屏幕上。下面的程序可以完成这个功能：

```
1:  #include <iostream.h>
2:  #include "matrix.h"
3:  void main()
```

```
4:  {
5:    static double aa[2][2]={{1.0,2.0},{3.0,4.0}};
6:    static double bb[2][2]={{2.0,2.0},{4.0,6.0}};
7:    matrix a(aa,2,2),b(bb,2,2);
8:    cout <<2.0* ~a*b+3.0;          // 这是此程序的核心，实现运算并输出到屏幕
9:  }
```

程序的第 1 行包含了一个 C++的流处理文件 iostream.h。注意，如果使用的是 VC++，则在第 1 行之前还必须加一行内容：#include "stdafx.h"，这是 VC++和 C++语言的不同之处。

第 2 行包含了文件 matrix.h，凡是要用到本章中提到的类的功能和函数的都必须包含它。

第 3 行为主程序的入口函数 main，所有 C++语言都从这个函数开始运行，被第 4 行和第 9 行的大括号{ }括起来的部分就是函数体。

第 5 行和第 6 行说明两个实数数组 aa 和 bb 分别存放矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的内容。

第 7 行则将这两个数组 aa 和 bb 包装到本书设计的 matrix 类型的类库变量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 中，指明这两个矩阵变量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的维数都是 2 行 2 列。

核心的计算在第 8 行，它完成了矩阵的计算并送到屏幕打印。在 C++语言中，cout 代表一个标准的输出设备，或称为输出流。如果有变量的内容想要送到屏幕打印，就将这个内容跟在两个小于号（<<）后面送到 cout 即可，无论是一段字符，还是一个整数或者实数变量、矩阵变量都可以。为了使用这个流的功能，第 1 行要包含 iostream.h 文件。

表达式 2.0*~$\boldsymbol{a}$*$\boldsymbol{b}$+3.0 完成所要求的计算 $2\boldsymbol{A}^{-1}\boldsymbol{B}+\boldsymbol{C}$，因为矩阵 $\boldsymbol{C}$ 是一个常数为 3 的矩阵，所以表达式后面只要简单地加上 3 就可以了，这种“数加”的算法是作者参照矩阵的数乘而自作主张定义的。~$\boldsymbol{a}$ 利用运算符~对 $\boldsymbol{a}$ 进行求逆，乘法使用*，这点同其他变量类型的乘法使用在形式上一致。

程序执行完毕后屏幕上显示：

```
matrix 2 2
1: 3 7
2: 5 3
```

这是一个矩阵变量输出的“标准”格式，这“标准”是作者想当然地定的。其中第一行以关键字 matrix 开头，后面紧跟着行数和列数，后面紧跟着一行接一行地显示数据，每一行以行号加冒号开头，后面跟着用空白字符隔开的该行各列的数据。因此我们算出了正确的结果 $\begin{pmatrix}3 & 7\\ 5 & 3\end{pmatrix}$。

其实，上面程序的重点是第 7 行的对矩阵类变量做初始化，以及第 8 行的表达式（说明矩阵怎样计算）。cout 作为 C++流在编写实用的软件时基本不用，因为现在要么倾向于用很好看的界面来显示数据，要么就是用自动控制系统，算出的数据又去控制其他的接口。第 7 行的初始化只是其中不常用的一种，因为实际的数据通常存放在数据文件中，或存放在数据库中，通过一个输入口来转换，很少直接写成内存数组的形式。后面将介绍其他的初始化方法。

这个例子为了简单只用了 2 行 2 列的数组，更大的数组（如几千个元素的数组）使用方法

也一样。想象一下，如果不用作者编写的矩阵类，而是自己完成矩阵求逆到矩阵相乘的各种算法，包括申请内存中转等一系列麻烦的事情，各种的循环套循环，将是多么麻烦的事！而现在有了作者编写的矩阵类，编写的效率大大提高，程序的质量也相当可靠。

1.1.2　矩阵类变量的初始化

在进行矩阵运算之前，首先要将各种数据包装到矩阵类的变量后面、简称为矩阵变量中去。总共有三类初始化矩阵类的变量的办法，一是利用内存数组或者指针，二是利用数据文件，三是分别对每个元素进行赋值。下面分别叙述。

1．利用内存数组或指针初始化矩阵变量

matrix 是作者定义的一个矩阵类，使用起来就像一般的 C 语言数据类型，如 int，double 那样。如语句

```
matrix a;
```

就初始化了一个叫做 ***a*** 的矩阵变量，但此矩阵变量在被赋值之前是个空矩阵。C++语言可以对每一个类编写各种构造函数，以适应各种不同的初始化需要。也就是在上面说明的矩阵变量右边加一对圆括号，在括号中间按照要求填入相应的数据。下面的程序就初始化了两个矩阵变量 ***a*** 和 ***b***。

```
static double aa[2][2]={{1.0,2.0},{3.0,4.0}};
static double bb[2][2]={{2.0,2.0},{4.0,6.0}};
matrix a(aa,2,2),b(bb,2,2);
```

这种构造函数需要三个变量，即一个实数数组变量或者指向实数的指针，后跟的两个数代表矩阵的行数和列数。注意行数和列数的乘积不可超过相应的实数数组的元素总数。其实，上面的数组变量即使是一维的数组也是可以的，将上面的程序改成：

```
static double aa[4]={1.0,2.0,3.0,4.0};
static double bb[4]={2.0,2.0,4.0,6.0};
matrix a(aa,2,2),b(bb,2,2);
```

也一样工作，而多维数组的顺序是先排完第一行的各列数字，然后再排第二行的各列数字，以此类推。

或者用一个指向实数的指针，申请一段内存，装入数据之后，再包装到 matrix 类的变量中。

2．利用数据文件进行矩阵变量的输入与输出

这里定义的矩阵数据文件格式是文本文件，可以用文本编辑器进行编写或者由其他的程序产生。文件的开始以关键字 matrix 打头，注意必须用小写，后面是空白字符（即空格或者制表符、回车换行符等），然后依次是矩阵的行数、空白字符、矩阵的列数、空白字符、各行数据。每一行以行号开头，后面紧跟冒号，再后面就是用空白字符隔开的该行数据。假设数据文件 test.dat 有如下的内容：

```
matrix 2 3
1: 1.2 3.3 -33
2: 0.2 -5 2.5
```

则语句

```
matrix a("test.dat");
```

将矩阵变量 ***a*** 初始化成内容为 $\begin{pmatrix} 1.2 & 3.3 & -33 \\ 0.2 & -5 & 2.5 \end{pmatrix}$ 的 2 行 3 列矩阵。

用 C++流也可以做到这一点，但要在程序源文件的开始加上以下语句。

```
#include <fstream.h>
```

下面的一段程序打开一个与文件 test.dat 连接的流，建立一个空的矩阵变量 ***a***，然后通过流将数据读入 ***a***：

```
ifstream if("est.dat");
matrix a;
if >> a;
```

这样的语句虽然比较啰唆，但在经常要使用一个矩阵变量来先后装载多个矩阵数据时，可能有用处。

矩阵变量中的内容也可以通过流送到相应的文件中去。下面的程序行将矩阵变量 ***a*** 中的内容送到 test1.dat 的文件中去：

```
ofstream of("test1.dat");
matrix a;
…(中间的运算使 a 中存放着某个运算结果)
of << a;
```

这里送出的数据可能是一个运算结果，可以通过其他的程序或者另外的文件流将数据再读回来。

其实，其他语言（如 VB 或者 Pascal）也可以通过这种数据文件格式与本书的 C++程序进行数据的传送。

3．直接对矩阵变量中的每个元素进行存取

语句

```
matrix a;
```

初始化一个空的矩阵，语句

```
matrix a(2,3);
```

初始化一个 2 行 3 列的矩阵，而语句

```
matrix a(5);
```

则初始化一个 5 行 1 列的矩阵，也就是一个列向量。

如果一个矩阵变量 ***a*** 原来是空的或者是 2 行 3 列的，怎样把它变成 4 行 5 列的呢？下面的语句可做到这一点：

```
a=matrix(4,5);
```

a 如果原来有内容，则原来的内容被冲掉了。

下面的语句将矩阵变量 ***a*** 的第 2 行第 3 列的值设为 4.7：

```
a.set(2,3,4.7);
```

其中的 a.set 调用了矩阵类的变量的一个成员函数，格式是变量名加下圆点加函数名，而调用方法则和一般的 C 语言没有什么不同。set 函数的头两个变量代表行号和列号，请注意在程序内部，行号和列号都是从 0 开始的。

为了方便一维列向量的值的设置，在取某行 0 列的元素时，列号可以省略，如下面的语句设置 ***a*** 的第 3 行第 0 列的数值为 4：

```
a.set(3,4.0);
```

下面的程序段将矩阵 ***a*** 的所有元素都设成 5：

```
for(size_t i=0; i<a.rownum; i++)
for(size_t j=0; j<a.colnum; j++)
a.set(i,j,5.0);
```

其中 rownum 和 colnum 是矩阵类的成员变量，用来存放矩阵类变量的行数和列数。

当然，上面只是示范怎样设置一个矩阵的所有元素。如果真的要将矩阵的每个元素都设成常数 5，那么应当使用下面的语句：

```
a = 5.0
```

这是因为 matrix 类已经设计重载了等号（=）或者叫赋值运算符的缘故，这种重载技术是 C++ 语言的重要技术。

也可以通过等号将一个矩阵变量的内容赋给另一个矩阵变量，如下面的语句将 ***b*** 的内容传送到 ***a*** 中，而 ***a*** 原来的内容（如果有）全都释放了。

```
matrix a,b;
…
a = b;
```

后面可以看到，matrix 类正是通过大量重载 C++语言的运算符，才使得数学演算变得简单了。

下面的程序行将矩阵变量 ***a*** 的第 2 行第 3 列的元素同 2 相乘后送到一实数变量 *x* 中：

```
x=a(2,3)*2;
```

这里矩阵变量似乎又像一个函数一样被调用，这是因为重载了函数运算符的缘故。这种方法比重载两个取数组元素操作符 a[2][3]的方案要好。而为了方便列向量的操作，后面的列号如果不写，就默认为 0，如下面的语句将 ***a*** 的第 5 行第 0 列的元素取到实数变量 *x* 中：

```
x = a(5);
```

1.1.3　矩阵的基本算法

本节介绍如何使用矩阵类完成各种常用的数学算法。在本节中，许多函数都会返回一个矩阵类型的引用。引用的实质是指向一个变量的指针，但为了方便程序员编程，无须进行像指针那样的复杂操作。

1. 赋值运算

矩阵类变量因为重载了赋值运算符而使矩阵的复制变得简单，假设有矩阵变量 ***a*** 和 ***b***，则语句

```
a=b;
```

将 ***b*** 的内容赋给 ***a***。

在实际应用中，更经常用到的是将一个能够返回矩阵或矩阵的引用的表达式赋给一个矩阵，如语句

```
a=b+c;
```

就将矩阵 ***b*** 和 ***c*** 相加后赋给矩阵变量 ***a***。

还可以临时构造一个新的矩阵变量赋给矩阵变量 ***a***，如下面的语句就不管 ***a*** 原来的内容如何，将 ***a*** 重新设为 30 行 20 列的矩阵：

```
a=matrix(30,20);
```

而语句

```
b=matrix(10);
```

则将 ***b*** 重新设为 10 行 1 列的矩阵，或者说 10 维的列向量。

如果保持一个矩阵的阶数不变，而将其所有的元素都设为同一个常数，可以在赋值运算的右侧设为实数或者实数表达式即可，如下面的语句将 ***a*** 的所有元素都设为 0：

```
a=0.0
```

而语句

```
a=1.5*4
```

将 ***a*** 的所有元素都设为 1.5×4，即 6。

这个办法比使用循环进行设置要好，且经常用在矩阵变量的初始化中，尤其是在一个矩阵的大部分元素都一样（比如都是零），只有少量的元素不同时，就可以用这种办法先将矩阵初始化为所有元素都是最常用的那个值，然后再对个别的不同元素进行赋值。

赋值运算都返回了矩阵的引用，因此又可以在一个表达式中结合其他算法，如语句

```
a=b=c=d;
```

将矩阵变量 ***a***、***b*** 和 ***c*** 的内容都设成和 ***d*** 一样。

2. 加法与减法

两个矩阵 ***A*** 与 ***B*** 如果要做加法或者减法，它们的行数和列数必须对应相等，即 A 的行数等于 B 的行数，A 的列数等于 B 的列数。假设矩阵 ***A*** 与 ***B*** 均为 m 行 n 列，

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

通常简写为 $\boldsymbol{A}=\{a_{ij}\}_{mn}$，$\boldsymbol{B}=\{b_{ij}\}_{mn}$。则它们的加法和减法定义为：

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

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

此外，由于在实际运算中经常出现一个矩阵加上另一个常数矩阵的情况，也就是每个矩阵元素加上同一个数构成新的矩阵，因此作者设计了“数加”的运算，类似于矩阵的数乘运算。假设 c 为一实常数，则矩阵 $\boldsymbol{A}+c$ 定义为：

$$\boldsymbol{A}+c=\begin{pmatrix} a_{11}+c & a_{12}+c & \cdots & a_{1n}+c \\ a_{21}+c & a_{22}+c & \cdots & a_{2n}+c \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+c & a_{m2}+c & \cdots & a_{mn}+c \end{pmatrix}$$

而矩阵减常数相当于加上此常数的负数，即 $\boldsymbol{A}-c=\boldsymbol{A}+(-c)$。

矩阵求负则是将一个矩阵的元素统统改为它的负值，即

$$-\boldsymbol{A}=\begin{pmatrix} -a_{11} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & -a_{22} & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{m1} & -a_{m2} & \cdots & -a_{mn} \end{pmatrix}$$

这样也可以定义常数加矩阵为 $c+\boldsymbol{A}=\boldsymbol{A}+c$，常数减矩阵为 $c-\boldsymbol{A}=c+(-\boldsymbol{A})$。

有关 matrix 实数矩阵类，已经通过重载加操作符+、减操作符–、加等操作符+=、减等操作符–=，实现了矩阵的相加、矩阵加常数、矩阵求负、矩阵相减、矩阵减常数的运算。

假设 ***a*** 和 ***b*** 均为矩阵变量，x 为一实数变量，表 1-1 列出了基本的与加减有关的表达式及对应程序所做的事和表达式的值。

表 1-1　matrix 类的加法与减法表

表达式	表达式的解释及运行结果
a*+*b	计算、产生并返回 ***a*** 与 ***b*** 的和矩阵
a*+=*b	使 ***a*** 的每一元素加上对应的 ***b*** 的元素，返回 ***a*** 的引用
b*+*x 或 ***x*+*b***	产生并返回 ***b*** 中的所有元素都加上实数 *x* 后产生的新矩阵
a*+=*x	使 ***a*** 中每一元素加上实数 *x*
***a*.neg()**	使 ***a*** 变为原来的负矩阵，返回 ***a*** 的引用
–*b*	产生并返回 ***b*** 的负矩阵，但 ***b*** 本身不变
a*–*b	产生并返回矩阵 ***a*** 减去矩阵 ***b*** 后的新矩阵
a*–=*b	使 ***a*** 中每一元素减去 ***b*** 中对应的元素，返回 ***a*** 的引用
a*–*x	产生并返回 ***a*** 中所有元素都减去实数 *x* 后产生的新矩阵

（续表）

表达式	表达式的解释及运行结果
x–***a***	产生并返回矩阵 ***a*** 求负后所有元素都加上 *x* 的新矩阵
a–=*x*	***a*** 中所有元素都减去实数 *x*，并返回 ***a*** 的引用

从表中可以看出，对于操作符的重载使矩阵运算的表达式变得简单。比如说双目运算符的重载，使得四个矩阵变量 ***a***，***b***，***c***，***d*** 相加用 ***a***+***b***+***c***+***d*** 表示，而如果用原来 C 语言的技术，则要编写一个函数，比如说 madd(double a[ ], double b[ ],int m,int n)，其中数组 ***a*** 和 ***b*** 存放两个矩阵的值，而 *m* 和 *n* 为行数和列数，则上面四个矩阵相加就变成

```
madd(a,madd(b,madd(c,d,m,n),m,n),m,n)
```

让人看起来极为费解又容易搞错。

3．乘法

矩阵类定义的乘法包括矩阵之间的相乘和数乘矩阵。

如果矩阵 ***A*** 乘矩阵 ***B***，***A*** 的列数必须等于 ***B*** 的行数。假设 ***A*** 为 *n*×*l* 阶矩阵，***B*** 为 *l*×*m* 阶矩阵，则它们的乘积 ***C***=***AB*** 为 *n*×*m* 阶矩阵，其中的元素 c_{ij} 按下式计算

$$c_{ij}=\sum_{k=1}^{l}a_{ik}b_{kj}\qquad(i=1,\cdots,n,\ \ j=1,\cdots,\ m)$$

而矩阵与一个数相乘，就是将矩阵的所有元素乘上这个数构成新的矩阵，也就是

$$c\boldsymbol{A}=\begin{pmatrix}ca_{11} & ca_{12} & \cdots & ca_{1m}\\ ca_{21} & ca_{22} & \cdots & ca_{2m}\\ \vdots & \vdots & \ddots & \vdots\\ ca_{n1} & ca_{n2} & \cdots & ca_{nm}\end{pmatrix}$$

假设 ***a*** 和 ***b*** 为矩阵变量，而 *x* 为双精度实数变量，表 1-2 列出了表示乘法的用法，以及重载除法操作符表示一个实数的倒数乘矩阵的用法。

表 1-2　matrix 类的乘法与除法表

表达式	表达式的解释及运行结果
a**b	计算、产生并返回 ***a*** 与 ***b*** 的乘积矩阵
a**=b	将 ***a*** 乘 ***b*** 的结果放于 ***a*** 中，返回 ***a*** 的引用
a**x* 或者 *xa***	实数 *x* 乘矩阵 ***a*** 返回新的矩阵
a**=x	矩阵 ***a*** 的每个元素乘实数 *x*，返回 ***a*** 的引用
a**/=x	矩阵 ***a*** 的每个元素除以实数 *x*，返回 ***a*** 的引用
a**/x	计算并返回实数 *x* 的倒数与 ***a*** 乘积的结果矩阵

1.1.4　解线性方程组

1．用高斯消元法解线性方程组

（1）方法和用法

线性方程组就是指三个矩阵 $\boldsymbol{A}$、$\boldsymbol{X}$ 和 $\boldsymbol{B}$ 满足关系 $\boldsymbol{AX}=\boldsymbol{B}$，而 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的内容已知，要求出 $\boldsymbol{X}$ 的值，以满足上面的等式。假设 $\boldsymbol{A}$ 为 $n\times n$ 方阵（称为系数矩阵），$\boldsymbol{X}$ 为 $n\times m$ 阶矩阵（由 m 个向量拼成），$\boldsymbol{B}$ 为 $n\times m$ 阶矩阵（称为常数矩阵）。当然，只有在矩阵 $\boldsymbol{A}$ 可逆的情况下方程才有唯一解，这时解为 $\boldsymbol{A}^{-1}\boldsymbol{B}$。

这里采用全选主元高斯-约当消元法（以下简称高斯消元法）来解线性方程组。先说消元法的基本原理。

消元法就是通过线性方程之间的加减法运算使一些变元消去。假设在操作中矩阵 $\boldsymbol{A}$ 与 $\boldsymbol{B}$ 的各个元素都不为零，则将 $\boldsymbol{A}$ 与 $\boldsymbol{B}$ 拼成一个矩阵（$\boldsymbol{A}|\boldsymbol{B}$），此矩阵被称为增广矩阵。

$$(\boldsymbol{A}\,|\,\boldsymbol{B})=\left(\begin{array}{ccc|ccc} a_{11} & \cdots & a_{1n} & b_{11} & \cdots & b_{1m} \\ a_{21} & \cdots & a_{2n} & b_{21} & \cdots & b_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_{n1} & \cdots & b_{nm} \end{array}\right)$$

那么，首先把第 1 行的元素统统除以左上角第一个元素 a_{11}，相当于第一个方程除以了一个常数，当然这个方程的解还是不变。而这么一来左上角第 1 个元素就变成了 1。再将第 2 行的所有元素都减去 a_{21} 倍的第 1 行，当然方程组的解还是不会变，但第 2 行第 1 列的元素 a_{21} 就被减成了 0，这样第 2 行就代表了一个少了一个元的方程，变成了 n–1 元方程。对第三行以后的方程都如法炮制，就是第 i 行的元素减去 a_{i1} 倍的第 1 行元素（i=2,3,⋯,n）。这样，除第 1 行外，第 2 行以后代表的方程组中的第一个变元 x_1 的系数全部变成了零，也就是说，除第一个方程以外的 n–1个方程构成了 n–1 元的线性方程组，这样就消去了一个元。这 n–1 元线性方程组的增广矩阵正好是变换后的第 2 行以后各行的第 2 行以后的各列的元素构成的子矩阵，称它为子增广矩阵。对这样一个子增广矩阵同样施行如上的手法，就逐渐地将方程的各个元都消掉，最后使增广矩阵的左边成为单位矩阵，这时右边就正好是方程的解，就是变成

$$(\boldsymbol{E}\,|\,\boldsymbol{X})=\left(\begin{array}{ccc|ccc} 1 & \cdots & 0 & x_{11} & \cdots & x_{1m} \\ 0 & \cdots & 0 & x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & x_{n1} & \cdots & x_{nm} \end{array}\right)$$

上面操作可能存在的问题是，在消元的第 k 步要将第 k 行元素统统除以对角线元素 a_{kk}，而如果 a_{kk} 为 0，计算机就会出错，如果它太小，计算机就会产生较大的计算误差。因此，在第 k 步做消元之前，首先将第 k 行第 k 列的右下角绝对值最大的元素找出来，然后进行行与列的交换，使之被换到第 k 行第 k 列，这样消元就可以进行了，而且计算的舍入误差会小。这个绝对值最大的元素，就被称为主元。但在行列调换的过程中，要用一个整数数组将调换情况记录下来，在解完后再按相反的顺序调换回去。