中国科学院研究生教学丛书

矩 阵 计 算

〔美〕G. H. 戈卢布
C. F. 范洛恩 著

袁亚湘等 译

科 学 出 版 社
2001

内 容 简 介

本书系统地介绍了矩阵计算的基本理论和方法.内容包括:矩阵乘法、矩阵分析、线性方程组、正交化和最小二乘法、特征值问题、Lanczos方法、矩阵函数及专题讨论等.书中的许多算法都有现成的软件包实现,每节后还附有习题,并有注释和大量参考文献.

本书可作为高等学校数学系高年级本科生和研究生教材,亦可作为计算数学和工程技术人员的参考用书.

图字:01-1999-2147

图书在版编目(CIP)数据

矩阵计算/〔美〕G.H.戈卢布(Golub,G.H.),〔美〕范洛恩(Van Loan,C.F.)著;袁亚湘等译.-北京:科学出版社,2001.8

(中国科学院研究生教学丛书/白春礼主编)

ISBN 7-03-008590-6

Ⅰ.矩… Ⅱ.①戈… ②范… ③袁… Ⅲ.矩阵-计算方法 Ⅳ.0241.6

中国版本图书馆CIP数据核字(2000)第62688号

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

科地亚印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2001年8月第 一 版 开本:850×1168 1/32

2001年8月第一次印刷 印张:26 1/2

印数:1—2 500 字数:697 000

定价:48.00元

(如有印装质量问题,我社负责调换〈新欣〉)

中译本前言

我很高兴我们的著作被译成中文出版.计算数学在当代中国正有很大的发展.这些成就是在已故冯康教授的领导下取得的,他的工作是开创性和播种性的.数值代数是至关重要的,毫无疑问在中国它已经很兴旺.中国高等院校坚实的数学训练,造就了许多优秀的青年数值分析专家,这使美国和其他西方国家受益匪浅.

我希望中译本能进一步增强中国读者对该领域的兴趣,并加强我们在该领域的联系与合作.

Gene H. Golub
2000.9.14

译者前言

Gene H. Golub 是美国 Stanford 大学教授，美国两院(美国科学院、美国工程院)院士，是当今世界上最著名的数值线性代数专家.他和 Charles F. Van Loan 合著的"Matrix Computations"是目前国际上数值线性代数方面最权威、最全面的一本专著.该书 1983 年出版，1989 年再版，本书译自 1996 年的第三版.

1989 年，我要在中国科学院研究生院讲授"数值线性代数"，就给 Golub 教授发了个 E-mail 表示希望能用他们的专著作为教材.他当即让 Johns Hopkins 大学出版社给我寄来了尚未正式出版的第二版的校样一本.多年来，这本书一直是我在研究生院讲"数值线性代数"的教材.

该书的翻译工作是由我和我们所三位研究生合作进行的，1～3 章由我自己翻译，其他的章节由三位研究生先译出一个初稿(叶军涛负责 4～6 章；朱宗武负责 7～8 章；聂家旺负责 9～12 章)，然后全书由我来统校.

该书原文(英文)写作优美，由于译者无论是英文还是中文水平都有限，难免会有不妥之处，欢迎广大读者批评指正.

本书的翻译和出版得到国家自然科学基金委和中国科学院研究生教材建设基金的支持，在此表示感谢.我还要感谢科学出版社的林鹏先生对此书出版所给予的支持.

中国科学院计算数学与科学工程计算研究所

袁亚湘

1999.9.4

第一章　矩阵乘法

研究矩阵计算的合适出发点是矩阵与矩阵的乘法.这一问题在数学上虽然简单,但从计算上来看却是十分丰富的.在§1.1中,我们看到矩阵相乘可以有好几种不同的方式.引进了分块矩阵,并将其用来刻画计算上的几种线性代数的"级".

如果一个矩阵具有某种结构,常常可加以利用.例如,一个对称矩阵只需一个一般矩阵一半的空间即可储存.在矩阵乘向量中,如果矩阵中有许多零元素,则可减少许多计算时间.这些问题将在§1.2中讨论.

在§1.3中定义了块矩阵记号.块矩阵是一个由矩阵组成的矩阵.这一概念无论是在理论上还是在实践中都是十分重要的.在理论方面,块矩阵使得重要的矩阵分解的证明十分简洁.这些分解是数值线性代数的基石.从计算的角度看,块算法中有大量的高性能计算机结构都善于进行的矩阵运算,因而是重要的.

这些新的结构要求算法设计者对存储通信与实际的计算量同等重视.科学计算的这一点在§1.4中阐明.在这一节将讨论,向量流水线计算的重要因素:向量长度,向量存取和向量再利用的级.

预备知识

熟悉 MATLAB 语言是重要的,可参阅 Pratap(1995)和 Van

Loan(1996)的教材.关于高性能矩阵计算的一本丰富的入门书是Dongarra,Duff,Sorersen 和 Vorst(1990).本章中与 LAPACK 相关的软件如下：

LAPACK： 一些一般运算		
_SCAL	$x \leftarrow \alpha x$	标乘
DOT	$u \leftarrow x^{T}y$	内积
_AXPY	$y \leftarrow \alpha x + y$	SAXPY
_GEMV	$y \leftarrow \alpha \boldsymbol{A}x + \beta y$	矩阵向量相乘
_GER	$\boldsymbol{A} \leftarrow \boldsymbol{A} + \alpha xy^{T}$	秩 1 修正
_GEMM	$c \leftarrow \alpha \boldsymbol{AB} + \beta c$	矩阵相乘

LAPACK： 一些对称运算		
_SYMV	$y \leftarrow \alpha \boldsymbol{A}x + \beta y$	矩阵向量相乘
_SPMV	$y \leftarrow \alpha \boldsymbol{A}x + \beta y$	矩阵向量相乘
_SYR	$\boldsymbol{A} \leftarrow \alpha xx^{T} + \boldsymbol{A}$	秩 1 修正
_SYR2	$\boldsymbol{A} \leftarrow \alpha xy^{T} + \alpha yx^{T} + \boldsymbol{A}$	秩 2 修正
_SYRK	$\boldsymbol{C} \leftarrow \alpha \boldsymbol{AA}^{T} + \beta \boldsymbol{C}$	秩 k 修正
_SYR2K	$\boldsymbol{C} \leftarrow \alpha \boldsymbol{AB}^{T} + \beta \boldsymbol{BA}^{T} + \beta \boldsymbol{C}$	秩 $2k$ 修正
_SYMM	$\boldsymbol{C} \leftarrow \alpha \boldsymbol{AB} + \beta \boldsymbol{C}$ 或 $(\alpha \boldsymbol{BA} + \beta \boldsymbol{C})$	对称/一般乘积

LAPACK： 一些带状/三角运算		
_GBMV	$y \leftarrow \alpha \boldsymbol{A}x + \beta y$	一般带状
_SBMV	$y \leftarrow \alpha \boldsymbol{A}x + \beta y$	对称带状
_TBMV	$x \leftarrow \alpha \boldsymbol{A}x$	三角阵
_TPMV	$x \leftarrow \alpha \boldsymbol{A}x$	三角阵(按列存)
_TRMM	$\boldsymbol{B} \leftarrow \alpha \boldsymbol{AB}$ 或 $(\alpha \boldsymbol{BA})$	三角/一般 乘积

§1.1 基本算法与记号

矩阵计算是基于线性代数运算的. 内积运算包括标量的加法和乘法. 矩阵向量相乘由内积组成,矩阵与矩阵相乘相当于一系列的矩阵向量相乘. 所有这些运算都可以用算法形式或者用线性代数的语言来描述. 这一节的主要任务是说明这两种表达方式如何互补. 我们将逐步引入一些记号以及让读者熟悉矩阵计算领域最根本的思维方式,讨论将围绕矩阵相乘,这种计算有几种不同的方式.

1.1.1 矩阵记号

$\mathbb{R}$ 表示实数集合,$\mathbb{R}^{m\times n}$ 表示所有 m 行 n 列矩阵组成的向量空间:

$$\boldsymbol{A}\in\mathbb{R}^{m\times n}\Leftrightarrow\boldsymbol{A}=(a_{ij})=\begin{bmatrix}a_{11} & \cdots & a_{1n}\\ \vdots & & \vdots\\ a_{m1} & \cdots & a_{mn}\end{bmatrix},\quad a_{ij}\in\mathbb{R}.$$

如果一个大写字母(如 $\boldsymbol{A},\boldsymbol{B},\boldsymbol{\Delta}$)表示一个矩阵,则带下标 ij 的对应的小写字母(如 $a_{ij},b_{ij},\delta_{ij}$)表示矩阵在$(i,j)$的元素. 适当时,我们也用$[\boldsymbol{A}]_{ij}$和$\boldsymbol{A}(i,j)$表示矩阵元素.

1.1.2 矩阵运算

基本矩阵运算包括转置($\mathbb{R}^{m\times n}\to\mathbb{R}^{n\times m}$):

$$\boldsymbol{C}=\boldsymbol{A}^{\mathrm{T}}\Rightarrow c_{ij}=a_{ji},$$

相加($\mathbb{R}^{m\times n}+\mathbb{R}^{m\times n}\to\mathbb{R}^{m\times n}$):

$$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}\Rightarrow c_{ij}=a_{ij}+b_{ij},$$

标量-矩阵相乘($\mathbb{R}\times\mathbb{R}^{m\times n}\to\mathbb{R}^{m\times n}$):

$$\boldsymbol{C}=\alpha\boldsymbol{A}\Rightarrow c_{ij}=\alpha a_{ij},$$

和矩阵-矩阵相乘($\mathbb{R}^{m\times p}\times\mathbb{R}^{p\times n}\to\mathbb{R}^{m\times n}$):

$$\boldsymbol{C}=\boldsymbol{A}\boldsymbol{B}\Rightarrow c_{ij}=\sum_{k=1}^{p}a_{ik}b_{kj}.$$

这些运算是矩阵计算的基本运算.

1.1.3 向量记号

$\mathbb{R}^n$ 表示 n 维实向量空间:

$$x \in \mathbb{R}^n \Leftrightarrow x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R}.$$

我们记 x_i 为 x 的第 i 个元素.根据文中需要,我们有时也用 $[x]_i$ 和 $x(i)$ 表示 x_i.

注意到 $\mathbb{R}^n$ 等于 $\mathbb{R}^{n\times 1}$,所以 $\mathbb{R}^n$ 中的元素是列向量.在另一方面,$\mathbb{R}^{1\times n}$ 中的元素是行向量:

$$x \in \mathbb{R}^{1\times n} \Leftrightarrow x = (x_1, \cdots, x_n).$$

如果 x 是一个列向量,则 $y = x^{\mathrm{T}}$ 是行向量.

1.1.4 向量运算

假定 $\alpha \in \mathbb{R}$,$x \in \mathbb{R}^n$ 和 $y \in \mathbb{R}^n$.基本向量运算包括标量-向量乘法:

$$z = \alpha x \Rightarrow z_i = \alpha x_i,$$

向量加法:

$$z = x + y \Rightarrow z_i = x_i + y_i,$$

点积(或称内积):

$$c = x^{\mathrm{T}} y \Rightarrow c = \sum_{i=1}^{n} x_i y_i,$$

向量相乘(或称 Hadamard 乘积):

$$z = x * y \Rightarrow z_i = x_i y_i.$$

另一个非常重要的运算是 saxpy,它具有修正形式:

$$y = ax + y \Rightarrow y_i = ax_i + y_i.$$

这里,符号"="用来表示赋值而不是数学上的等号.此运算修正向量 y,它的名称 saxpy 是在 LAPACK 中所用的.LAPACK 是一个

将本书中的许多算法都实现了的软件包. saxpy 是标量 a 乘 x 加 y(scalar a x plus y)的缩写.

1.1.5 点积和 Saxpy 计算

我们用 MATLAB 语言的形式来描述算法,MATLAB 是一种对矩阵计算非常理想的有效的交互系统. 在这一章我们将逐步引入 MATLAB 的记号,首先我们给出一个计算内积的算法.

算法 1.1.1(内积) 给出 $x, y \in \mathbb{R}^n$,此算法计算内积 $c = x^{\mathrm{T}}y$.

```
c = 0
for i = 1 : n
    c = c + x(i)y(i)
end
```

两个向量的内积涉及到 n 次乘法和 n 次加法,所以它是 $O(n)$运算,即工作量是维数的线性函数. saxpy 也是 $O(n)$运算,但它的返回值是一个向量而不是标量.

算法 1.1.2(saxpy) 给定 $x, y \in \mathbb{R}^n$ 和 $a \in \mathbb{R}$,此算法用 $ax + y$ 覆盖 y.

```
for i = 1:n
    y(i) = ax(i) + y(i)
end
```

必须强调的是本书所给出的算法是关键的计算思想的框架而不是成品软件.

1.1.6 矩阵-向量乘法和 Gaxpy

假定 $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ 和 $y \in \mathbb{R}^m$,我们需要计算

$$y = \boldsymbol{A}x + y.$$

这是广义的 saxpy,称之为 gaxpy. 此计算的常规方式是每次修正一个分量,即

$$y_i = \sum_{j=1}^{n} a_{ij}x_j + y_i, \qquad i = 1:m.$$

这就是如下的算法：

算法 1.1.3(Gaxpy:行型)　设 $A \in \mathbb{R}^{m\times n}$，$x \in \mathbb{R}^n$ 和 $y \in \mathbb{R}^m$，本算法用 $Ax + y$ 覆盖 y.

```
for  i = 1 : m
     for  j = 1 : n
          y(i) = A(i,j)x(j) + y(i)
     end
end
```

Ax 可表示为 A 的列向量的线性组合，如

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}7\\ 8\end{bmatrix} = \begin{bmatrix}1\times 7 + 2\times 8\\ 3\times 7 + 4\times 8\\ 5\times 7 + 6\times 8\end{bmatrix}$$

$$= 7\begin{bmatrix}1\\ 3\\ 5\end{bmatrix} + 8\begin{bmatrix}2\\ 4\\ 6\end{bmatrix} = \begin{bmatrix}23\\ 53\\ 83\end{bmatrix}.$$

这一计算方式导致如下算法：

算法 1.1.4(Gaxpy:列型)　设 $A \in \mathbb{R}^{m\times n}$，$x \in \mathbb{R}^n$ 和 $y \in \mathbb{R}^m$. 本算法用 $Ax + y$ 覆盖 y.

```
for  j = 1 : n
     for  i = 1 : m
          y(i) = A(i,j)x(j) + y(i)
     end
end
```

注意到这两个 Gaxpy 算法的内层循环都是 saxpy 运算. 列型的算法是从向量层次重新理解矩阵与向量相乘所导出的，它也可从行型算法简单地交换求和顺序而得到. 在矩阵计算中，将求和顺序的交换与对应的线性代数联系起来是重要的.

1.1.7 矩阵分划成行和列

算法 1.1.3 和算法 1.1.4 是分别按行和按列用到 A 的数据为了更清楚地说明这一点我们引进矩阵分划的概念.

从行来看,一个矩阵是由行向量堆成的:

$$A \in \mathbb{R}^{m\times n} \Leftrightarrow A = \begin{bmatrix} r_1^{\mathrm{T}} \\ \vdots \\ r_m^{\mathrm{T}} \end{bmatrix}, \quad r_k \in \mathbb{R}^n. \tag{1.1.1}$$

这称之为 A 的行分划.所以,当我们行分划

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

时,我们就把 A 看成是由行向量

$$r_1^{\mathrm{T}} = [1 \quad 2], \quad r_2^{\mathrm{T}} = [3 \quad 4], \text{和 } r_3^{\mathrm{T}} = [5 \quad 6]$$

组成.利用行分划(1.1.1),算法 1.1.3 可表示成

for $i = 1 : m$

$\quad y_i = r_i^{\mathrm{T}} x + y_i$

end

另一方面,矩阵也是由列向量组成的:

$$A \in \mathbb{R}^{m\times n} \Leftrightarrow A = [c_1, \cdots, c_n], \ c_k \in \mathbb{R}^m. \tag{1.1.2}$$

我们把这称为 A 的列分划.在上面的 3×2 的例子中,c_1 和 c_2 就分别是 A 的第一列和第二列:

$$c_1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \qquad c_2 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

利用(1.1.2),我们看到算法 1.1.4 是一个用到 A 的列向量的 saxpy

for $j = 1 : n$

$\quad y = x_j c_j + y$

end

在此 y 可看成重复 saxpy 修正的连续求和.

1.1.8 冒号记号

表示矩阵的一行或一列的一种简便方式是“冒号”记号. 如果 $\boldsymbol{A}\in\mathbb{R}^{m\times n}$,则 $A(k,:)$表示 $\boldsymbol{A}$ 的第 k 行,即

$$A(k,:)=[a_{k1},\cdots,a_{kn}].$$

第 k 列表示为

$$A(:,k)=\begin{bmatrix}a_{1k}\\ \vdots\\ a_{mk}\end{bmatrix}.$$

有了这些记号,我们可将算法 1.1.3 和算法 1.1.4 分别改写成

for $i=1:m$

$\quad y(i)=A(i,:)x+y(i)$

end

和

for $j=1:n$

$\quad y=x(j)A(:,j)+y$

end

利用冒号记号,我们可省略迭代细节. 因而我们可从向量层次来考虑以及把注意力集中在大的计算问题上.

1.1.9 外积修正

作为冒号记号的初步应用,我们用它来了解外积修正

$$\boldsymbol{A}=\boldsymbol{A}+xy^{\mathrm{T}},\ \boldsymbol{A}\in\mathbb{R}^{m\times n},\ x\in\mathbb{R}^{m},\ y\in\mathbb{R}^{n}.$$

外积算子 xy^{T} 由于是两个“细长”矩阵的乘积而“看上去滑稽”. 但它是完全合法,因为左边矩阵 x 的列数与右边矩阵 y 的行数相等. 例如

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}4 & 5\end{bmatrix}=\begin{bmatrix}4 & 5\\8 & 10\\12 & 15\end{bmatrix}.$$

外积修正的元素可表述为

for $i = 1:m$

for $j = 1:n$

$a_{ij} = a_{ij} + x_i y_j$

end

end

对 j 的循环是将 y^{T} 的倍数加到 $\boldsymbol{A}$ 的第 i 行，即

for $i = 1:n$

$A(i,:) = A(i,:) + x(i)y^{\mathrm{T}}$

end

另一方面，如果我们把对 i 的循环换到里层，则是将 x 的倍数加到 $\boldsymbol{A}$ 的第 j 列：

for $j = 1:n$

$A(:,j) = A(:,j) + y(j)x$

end

注意这两个外积修正算法都是由一组 saxpy 修正所组成.

1.1.10 矩阵-矩阵相乘

考虑 2×2 矩阵与矩阵相乘 $\boldsymbol{AB}$. 在内积形式下，每个元素可由一点积计算：

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}=\begin{bmatrix}1\times 5+2\times 7 & 1\times 6+2\times 8\\3\times 5+4\times 7 & 3\times 6+4\times 8\end{bmatrix}.$$

在 saxpy 形式下，乘积的每列可看成是 $\boldsymbol{A}$ 的列之线性组合：

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}=\left[5\begin{bmatrix}1\\3\end{bmatrix}+7\begin{bmatrix}2\\4\end{bmatrix},\quad 6\begin{bmatrix}1\\3\end{bmatrix}+8\begin{bmatrix}2\\4\end{bmatrix}\right].$$

最后，在外积形式下，矩阵相乘可看成是一组外积的和：

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}=\begin{bmatrix}1\\ 3\end{bmatrix}\begin{bmatrix}5 & 6\end{bmatrix}+\begin{bmatrix}2\\ 4\end{bmatrix}\begin{bmatrix}7 & 8\end{bmatrix}.$$

这几种矩阵相乘的形式虽然在数学上等价,但由于内存通信的不同,在计算上的表现可能是不同的.这一点将在§1.4中进一步讨论.现在,我们详细给出矩阵相乘的上述三种形式,这将有助于我们加深冒号记号的概念以及习惯从不同线性代数的层次上来思考.

1.1.11 标量描述

为讨论集中,我们考虑如下矩阵相乘的修正公式:

$$\boldsymbol{C}=\boldsymbol{AB}+\boldsymbol{C},\qquad \boldsymbol{A}\in\mathbb{R}^{m\times p},\boldsymbol{B}\in\mathbb{R}^{p\times n},\boldsymbol{C}\in\mathbb{R}^{m\times n}.$$

首先是熟悉的三层循环算法:

算法 1.1.5(矩阵乘法:*ijk* 形式) 给出 $\boldsymbol{A}\in\mathbb{R}^{m\times p},\boldsymbol{B}\in\mathbb{R}^{p\times n}$ 和 $\boldsymbol{C}\in\mathbb{R}^{m\times n}$,本算法用 $\boldsymbol{AB}+\boldsymbol{C}$ 覆盖 $\boldsymbol{C}$.

```
for i = 1 : m
    for j = 1 : n
        for k = 1 : p
            C(i,j) = A(i,k)B(k,j) + C(i,j)
        end
    end
end
```

由于我们把 $\boldsymbol{C}$(以及 $\boldsymbol{A}$)的行标为 i,$\boldsymbol{C}$(以及 $\boldsymbol{B}$)的列标为 j,求和的下标记为 k,所以这是一个 *ijk* 形式.

我们考虑修正公式 $\boldsymbol{C}=\boldsymbol{AB}+\boldsymbol{C}$ 而不是 $\boldsymbol{C}=\boldsymbol{AB}$ 有两个理由:一是不必去为 $\boldsymbol{C}=0$ 的初始化操心;二是在实际中 $\boldsymbol{C}=\boldsymbol{AB}+\boldsymbol{C}$ 出现得更频繁.

矩阵乘法中三个求和循环可任意排序,从而一共有 $3!=6$ 种形式.所以,

```
for j = 1 : n
    for k = 1 : p
```

```
        for  i = 1 : m
            C(i,j) = A(i,k)B(k,j) + C(i,j)
        end
    end
end
```

是 *jki* 形式. 这 6 种可能性(*ijk*, *jik*, *ikj*, *jki*, *kij*, *kji*)的任何一个都对应一内层运算(点积或 saxpy)而且具有自己的数据流动模式. 例如,在 *ijk* 形式,内层运算是点积,数据用到的是 $\boldsymbol{A}$ 的行和 $\boldsymbol{B}$ 的列. 而在 *jki* 形式下内层运算是 saxpy,数据是 $\boldsymbol{C}$ 的列和 $\boldsymbol{A}$ 的列. 这些性质以及把中层与内层连在一起考虑是代表何种运算都归纳在表 1.1.1 中. 每种形式的浮点运算次数都一样,但对 $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ 数据的存取却是不同的.

表 1.1.1 矩阵乘法:循环排序及特性

循环顺序	内层循环	中层循环	内层数据存取
ijk	点积	向量乘矩阵	$\boldsymbol{A}$ 的行, $\boldsymbol{B}$ 的列
jik	点积	矩阵乘向量	$\boldsymbol{A}$ 的行, $\boldsymbol{B}$ 的列
ikj	saxpy	行的 gaxpy	$\boldsymbol{B}$ 的行, $\boldsymbol{C}$ 的行
jki	saxpy	列的 gaxpy	$\boldsymbol{A}$ 的列, $\boldsymbol{C}$ 的列
kij	saxpy	行的外积	$\boldsymbol{B}$ 的行, $\boldsymbol{C}$ 的行
kji	saxpy	列的外积	$\boldsymbol{A}$ 的列, $\boldsymbol{C}$ 的列

1.1.12 点积形式

通常的矩阵乘法过程把 $\boldsymbol{AB}$ 视为用点积计算的数组,该数组可按从左到右,从上到下的次序逐个计算. 这就是算法 1.1.5 所用到的思想. 利用冒号记号我们可突出其点积计算:

算法 1.1.6(矩阵乘法:点积形式) 给出 $\boldsymbol{A}\in\mathbb{R}^{m\times p}$, $\boldsymbol{B}\in\mathbb{R}^{p\times n}$ 和 $\boldsymbol{C}\in\mathbb{R}^{m\times n}$,本算法用 $\boldsymbol{AB}+\boldsymbol{C}$ 覆盖 $\boldsymbol{C}$.

```
for  i = 1 : m
     for  j = 1 : n
          C(i,j) = A(i, :)B(:,j) + C(i,j)
     end
end
```

利用矩阵分划语言,记

$$\boldsymbol{A} = \begin{bmatrix} a_1^{\mathrm{T}} \\ \vdots \\ a_m^{\mathrm{T}} \end{bmatrix}, \qquad a_k \in \ ^{p}$$

和

$$\boldsymbol{B} = [b_1, \cdots, b_k], \qquad b_k \in \ ^{p},$$

则算法 1.1.6 可表示为

```
for  i = 1 : m
     for  j = 1 : n
          c_ij = a_i^T b_j + c_ij
     end
end
```

注意到 j 循环的"任务"是计算修正公式的第 i 行,为强调这一点,我们可写成

```
for  i = 1 : m
     c_i^T = a_i^T B + c_i^T
end
```

这里

$$\boldsymbol{C} = \begin{bmatrix} c_1^{\mathrm{T}} \\ \vdots \\ c_m^{\mathrm{T}} \end{bmatrix}$$

是 $\boldsymbol{C}$ 的行分划.用冒号记号来表示这同一运算,我们可写成

```
for  i = 1 : n
     C(i, :) = A(i, :)B + C(i, :)
```

end

无论从哪种形式我们都看到 ijk 形式的内部的两层循环是基于行的 gaxpy 运算.

1.1.13 Saxpy 形式

假定 $\boldsymbol{A}$ 和 $\boldsymbol{C}$ 的列分划为

$$\boldsymbol{A} = [a_1, \cdots, a_p], \qquad a_j \in \quad ^m,$$
$$\boldsymbol{C} = [c_1, \cdots, c_n], \qquad c_j \in \quad ^m.$$

比较 $\boldsymbol{C} = \boldsymbol{AB} + \boldsymbol{C}$ 两边第 j 列可知

$$c_j = \sum_{k=1}^{p} b_{kj} a_k + c_j, \quad j = 1 : n$$

这些向量求和可通过一系列 saxpy 运算集中起来.

算法 1.1.7(矩阵乘法:saxpy 形式) 给出 $\boldsymbol{A} \in \ ^{m\times p}$, $\boldsymbol{B} \in \ ^{p\times n}$,和 $\boldsymbol{C} \in \ ^{m\times n}$,本算法用 $\boldsymbol{AB} + \boldsymbol{C}$ 覆盖 $\boldsymbol{C}$.

```
for  j = 1 : n
     for  k = 1 : p
          C(:,j) = A(:,k)B(k,j) + C(:,j)
     end
end
```

注意到 k 循环实质上是一 gaxpy 运算:

```
for  j = 1 : n
     C(:,j) = AB(:,j) + C(:,j)
end
```

1.1.14 外积形式

考虑算法 1.1.5 的 kji 形式:

```
for  k = 1 : p
     for  j = 1 : n
          for  i = 1 : m
               C(i,j) = A(i,k)B(k,j) + C(i,j)
```