符号计算程序分析

——在线性代数、矩阵论中的应用

智慧来 智东杰 著

科学出版社

符号计算程序分析

——在线性代数、矩阵论中的应用

智慧来　智东杰　著

科学出版社

北京

内 容 简 介

随着计算机技术的飞速发展，计算机代数系统已经广泛地应用于科研、教学以及工程技术中，如著名的 Maple、Mathematica 和 Matlab 等。它们在线性代数及矩阵论的教学中应用研究的较少，不够深入，本书对此进行比较深入的研究。全书共 8 章，分别介绍了 n 阶行列式的计算，矩阵及其运算，解实矩阵方程，线性方程组，矩阵的 Doolittle 和 Crout 分解，复矩阵乘法，计算复数行列式及求解复矩阵方程，指针在符号行列式计算中的应用。

本书适合具有线性代数知识和 C 语言程序设计基础的大学生及讲授线性代数课的教师阅读，也适合研究符号计算的科研人员和工程技术人员阅读。

图书在版编目(CIP)数据

符号计算程序分析：在线性代数、矩阵论中的应用/智慧来，智东杰著. —北京：科学出版社，2012
ISBN 978-7-03-034149-5

Ⅰ. 符… Ⅱ. ①智… ②智… Ⅲ. 数值计算-应用软件 Ⅳ. O245

中国版本图书馆 CIP 数据核字(2012)第 080572 号

责任编辑：魏英杰 杨向萍 / 责任校对：朱光兰
责任印制：张 倩 / 封面设计：陈 敬

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

骏 走 印 刷 厂 印刷

科学出版社发行 各地新华书店经销

*

2012 年 5 月第 一 版 开本：B5(720×1000)
2012 年 5 月第一次印刷 印张：14 3/4
字数：284 000

定价：45.00 元

(如有印装质量问题，我社负责调换)

前　言

符号计算与代数计算的学科称为计算机代数，或数学机械化。本书所有计算程序的特点是对符号按确定的规则进行演算，并且计算过程都是精确的，即符号计算。

特点之一：本书所提供的C程序经编译系统编译后产生的可执行文件离开编译环境照样可以使用，而不像语言教科书中提供的程序，只能在环境下运行。

特点之二：举一反三。学会编写一个元素为整数的行列式的计算程序设计方法，就会联想到多个方法。例如，元素为有理数的行列式的计算方法，元素为复数（实部和虚部都是整数）行列式的计算方法，元素为复分数（实部和虚部均为有理数即分数）乃至更复杂的行列式，元素为文字，元素为一元多项式的行列式的计算程序。学会用数组存储元素到学会用指针存储元素；学会用一维数组来表示二维数组，等等。

本书前5章为与整数和有理数（分数）有关的计算程序；第6章和第7章为复数有关的计算程序；第8章为文字方面的程序，部分程序国外用DNA计算才能实现。

按照人们的学习习惯，本书分8章介绍：第1章给出了全排列及其逆序数的计算程序、按n阶行列式的定义开发的计算程序、行列式按行（列）展开的程序、克莱姆法则的计算程序、计算分数行列式、分数克莱姆法则；第2章给出矩阵乘法程序、矩阵的转置、逆阵、分数矩阵乘法计算程序；第3章介绍计算矩阵方程$AX=B$、计算矩阵方程$XA=B$、计算矩阵方程$AXB=C$；第4章给出了解非齐次线性方程组程序；第5章给出了矩阵的Doolittle和Crout分解；第6章给出了复矩阵乘法，如元素实部、虚部均为整形数的复矩阵程序、两矩阵元素的实部与虚部均是有理数、一个矩阵元素的实部与虚部均是整数，另一矩阵元素的实部与虚部均是有理数计算矩阵乘积；第7章给出了计算复数行列式、计算复数余子式、计算复分数行列式、计算复分数余子式、复克莱姆法则、解复数矩阵方程$AX=B$的程序；第8章给出了用指针在计算行列式的应用的程序、简单的符号行列式计算、使用指针计算符号行列式程序、元素为字母行列式程序，最后研究了使用指针计算元素形如$f(x)=\sum_{i=0}^{n}a_ix^{n-i}$的符号行列式的程序。

本书第 1 章、第 7 章及第 8 章由智东杰完成，第 2 章～第 6 章由智慧来完成。智东杰对全书统稿。

由于作者水平有限，书中难免有不妥之处，敬请广大读者提出宝贵意见。

智慧来　智东杰

2012 年 1 月于河南理工大学计算机科学与技术学院

目　录

前言

第 1 章　*n* 阶行列式的计算 …… 1

1.1　全排列及其逆序数的计算程序 …… 1

1.1.1　基本概念 …… 1

1.1.2　计算程序 …… 1

1.2　按 *n* 阶行列式的定义开发的计算程序 …… 4

1.2.1　行列式的概念 …… 4

1.2.2　行列式计算程序实现方法分析 …… 4

1.2.3　计算程序 …… 5

1.3　行列式按行(列)展开的程序 …… 9

1.3.1　按行(列)展开概念 …… 9

1.3.2　计算行列式及余子式的 C 程序 …… 9

1.3.3　行列式按行(列)展开的程序 …… 16

1.4　克莱姆法则的计算程序 …… 19

1.4.1　线性方程的方程组概念 …… 19

1.4.2　克莱姆法则计算程序 …… 20

1.5　计算分数行列式 …… 26

1.5.1　分数行列式例 …… 26

1.5.2　计算程序 …… 26

1.6　分数克莱姆法则 …… 33

1.6.1　分数计算程序 …… 33

1.6.2　部分算法分析 …… 37

1.6.3　分数克莱姆法则运行实例 …… 38

第 2 章　矩阵及其运算 …… 45

2.1　矩阵的概念 …… 45

2.2　矩阵乘法 …… 45

2.3　矩阵的转置 …… 48

2.4　逆阵 …… 51

2.5　元素为分数的矩阵乘法 …… 60

2.5.1　$c[i][j]$的算法分析 …… 60

2.5.2 分数矩阵乘法计算程序 …… 61
第 3 章 解实矩阵方程 …… 69
3.1 计算矩阵方程 $AX=B$ …… 69
3.1.1 计算矩阵方程 $AX=B$ 技术分析 …… 69
3.1.2 存储空间 …… 69
3.1.3 程序中二维数组和主要函数 …… 69
3.1.4 计算程序 …… 69
3.2 计算矩阵方程 $XA=B$ …… 83
3.2.1 计算矩阵方程 $XA=B$ 技术分析 …… 83
3.2.2 存储空间 …… 83
3.2.3 程序中二维数组和主要函数 …… 83
3.2.4 计算程序 …… 83
3.3 计算矩阵方程 $AXB=C$ …… 91
3.3.1 计算矩阵方程 $AXB=C$ 技术分析 …… 91
3.3.2 存储空间 …… 92
3.3.3 程序中二维数组和主要函数 …… 92
3.3.4 计算程序 …… 92
第 4 章 线性方程组 …… 106
4.1 非齐次线性方程组概述 …… 106
4.2 程序设计 …… 106
4.2.1 Turbo C 2.0 程序 …… 106
4.2.2 Visual C++ 6.0 程序 …… 114
第 5 章 矩阵的 Doolittle 和 Crout 分解 …… 121
5.1 Doolittle 分解与 Crout 分解概述 …… 121
5.2 Doolittle 分解程序及运行 …… 123
5.2.1 Turbo C 2.0 程序 …… 123
5.2.2 Visual C++6.0 程序 …… 126
5.3 Crout 分解计算程序及运行 …… 128
5.3.1 Turbo C 2.0 分解程序及示例 …… 128
5.3.2 Visual C++6.0 程序 …… 132
第 6 章 复矩阵乘法 …… 136
6.1 复矩阵乘法概述 …… 136
6.2 元素实部、虚部均为整形数的复矩阵 …… 136
6.2.1 Turbo C 2.0 程序 …… 136
6.2.2 计算实例 …… 139

6.3　两矩阵元素的实部与虚部均是有理数 …… 139
6.3.1　复分数矩阵乘法程序 …… 139
6.3.2　运行示例 …… 143
6.4　一个矩阵元素的实部与虚部均是整数，另一矩阵元素的实部与虚部均是有理数计算矩阵乘积 …… 144
6.4.1　分析与设置数据类型 …… 145
6.4.2　Turbo C 2.0 矩阵乘法程序(1) …… 145
6.4.3　Turbo C 2.0 矩阵乘法程序(2) …… 152
第 7 章　计算复数行列式及求解复矩阵方程 $AX=B$ …… 159
7.1　计算复数行列式 …… 159
7.1.1　复数行列式简述 …… 159
7.1.2　计算程序 …… 159
7.1.3　程序运行示例 …… 162
7.2　计算复数余子式 …… 162
7.2.1　复数余子式简述 …… 162
7.2.2　Turbo C 2.0 计算程序 …… 163
7.2.3　程序运行示例 …… 165
7.3　计算复分数行列式 …… 166
7.3.1　复分数行列式简述 …… 166
7.3.2　复分数行列式计算程序 …… 167
7.3.3　复分数行列式程序运行示例 …… 171
7.4　计算复分数余子式 …… 172
7.4.1　复分数余子式简述 …… 172
7.4.2　复分数余子式 Turbo C 2.0 计算程序 …… 173
7.5　复克莱姆法则 …… 179
7.5.1　复数域上的克莱姆法则简述 …… 180
7.5.2　复数域上的克莱姆法则程序设计 …… 180
7.5.3　复数域上的克莱姆法则 Turbo C 2.0 计算程序 …… 181
7.5.4　复数域上的克莱姆法则 Turbo C 2.0 程序运行示例 …… 187
7.6　解复数矩阵方程 $AX=B$ 的程序 …… 188
第 8 章　指针在符号行列式计算中的应用 …… 198
8.1　计算行列式的应用 …… 198
8.1.1　理论上完美无缺的 Turbo C 2.0 程序 …… 198
8.1.2　程序运行实例 …… 200
8.2　简单的符号行列式计算 …… 200

8.2.1 符号行列式 Turbo C 2.0 计算程序 …… 201
8.2.2 符号行列式 Visual C++6.0 程序 …… 205
8.3 使用指针计算简单的符号行列式 …… 209
8.3.1 简单说明 …… 209
8.3.2 使用指针计算符号行列式 Turbo C 2.0 程序 …… 209
8.4 字母行列式 …… 214
8.5 使用指针计算元素形如 $f(x)=\sum_{i=0}^{n}a_ix^{n-i}$ 的符号行列式 …… 216
8.5.1 程序组成 …… 216
8.5.2 使用指针计算元素一元多项式的符号行列式的 Turbo C 2.0 程序…… 217
参考文献…… 224

第 1 章　n 阶行列式的计算

1.1　全排列及其逆序数的计算程序

1.1.1　基本概念

在数学中，把考察的对象，叫做元素。

把 n 个不同的元素排成一列，叫做这 n 个元素的全排列(也简称排列)。

对于 n 个不同的元素，我们规定各元素之间有一个标准次序，于是在这 n 个元素的任一排列中，当某两个元素的先后次序与标准次序不同时，就说有 1 个逆序。一个排列中所有逆序的总数叫做这个排列的逆序数。

逆序数为奇数的排列叫做奇排列，逆序数为偶数的排列叫做偶排列。

对于计算排列逆序数，不失一般性，不妨设 n 个元素为 1～n 这 n 自然数，并规定由小到大为标准次序。设

$$p_1 p_2 \cdots p_n$$

为这 n 个自然数的一个排列，考虑元素 $p_i(i=1,2,\cdots,n)$，如果比 p_i 大且排在 p_i 前面的元素有 t_i 个，就说 p_i 这个元素的逆序数是 t_i，全体元素的逆序数之和为

$$t = t_1 + t_2 + \cdots + t_n = \sum_{i=1}^{n} t_i$$

即是这个排列的逆序数。

用 $p_j - p_i > 0$ 表示 p_j 排在 p_i 前面且比 p_i 大这一事实。排列的逆序数可表示为

$$t = \sum_{i=1}^{n} \sum_{j=1}^{i} g(i,j), \quad g(i,j) = \begin{cases} 1 & p_j > p_i \\ 0 & p_j \leqslant p_i \end{cases}, \quad i = 1,2,\cdots,n; j = 1,2,\cdots,i$$

在 C(C++)语言中表示为

$$t = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} g(i,j), \quad g(i,j) = \begin{cases} 1 & p_j > p_i \\ 0 & p_j \leqslant p_i \end{cases},$$
$$i = 0,1,\cdots,n-1; j = 0,1,\cdots,i-1$$

1.1.2　计算程序

Turbo C 2.0 程序

```
/*Inverse.c */
#include "conio.h"
main( )
{
  #define N 10                          /*定义 N 是 10 */
  int i,j,n,t,s,a[N];
  printf("n=");scanf("%d",&n);  /*n 是排列的长度,小于或等于 N */
  for(i=0;i<n;i++)
{
  printf("a[%d]=",i);scanf("%d",&a[i]);/*a[i]是排列中的元素,*/
                                        /*i=0,1,…,n-1*/
  }
  t=0;                                  /*t 代表排列的逆序数,初始化为 0 */
  for(i=0;i<n;i++)
  {
    s=0;                                /*s 是元素的 a[i]的逆序数*/
    for(j=0;j<i;j++)
      if(a[j]>a[i]) s++;                /*或 a[i]<a[j]存在 1 个逆序,s 加 1*/
    printf("%d inverse =%d\n",a[i],s);
    t=t+s;
  }
  printf("t=%d\n",t);
  printf("Press any key to end.\n");
  getch( );
}
```

求排列 43152 的逆序数,答案是 t=6。运算过程如下:

```
n=5
a[0]=4
a[1]=3
a[2]=1
a[3]=5
a[4]=2
4 inverse =0
3 inverse =1
1 inverse =2
```

```
5 inverse =0
2 inverse =3
t=6
Press any key to end.
```

Visual C++ 6.0 程序

```
/*Inverse.cpp  求逆序数 */
#include <iostream.h>
void main( )
{
  const int N=10;                    /*定义 N 是 10 */
  int i,j,n,t,s,a[N];
  cout<<"n=";cin>>n;                 /*n 是排列的长度,小于或等于 N */
  for(i=0;i<n;i++)
  {
    cout<<"a["<<i<<"]=";cin >>a[i];/*a[i]是排列中的元素,i=0,*/
                                     /*1,2,…,n-1*/
  }
  t=0;                               /*t 代表排列的逆序数,初始化为 0 */
  for(i=0;i<n;i++)
  {
    s=0;                             /*s 是元素的 a[i]的逆序数*/
    for(j=0;j<i;j++)
      if(a[j]>a[i]) s++;             /*或 a[i]<a[j]存在 1 个逆序,s 加 1*/
    cout << a[i] <<" inverse =" <<s<<endl;
    t=t+s;
  }
  cout <<"t="<<t<<endl;
}
```

求排列 32514 的逆序数,答案是 t=5。

```
n=5
a[0]=3
a[1]=2
a[2]=5
a[3]=1
a[4]=4
```

```
3  inverse =0
2  inverse =1
5  inverse =0
1  inverse =3
4  inverse =1
t=5
```

1.2 按 n 阶行列式的定义开发的计算程序

1.2.1 行列式的概念

定义 设有 n^2 个数，排成 n 行 n 列的表，即

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{matrix}$$

作出表中位于不同行不同列的 n 个数的乘积，并冠以符号 $(-1)^t$，得到的项为

$$(-1)^t a_{1p_1} a_{2p_2} \cdots a_{np_n} \tag{1.1}$$

其中，$p_1, p_2, \cdots, p_n$ 为自然数 $1, 2, \cdots, n$ 的一个排列；t 为这个排列的逆序数。

由于这样的排列共有 $n!$ 个，因而形如式(1.1)的项共有 $n!$ 项。这 $n!$ 项的代数和为

$$\sum (-1)^t a_{1p_1} a_{2p_2} \cdots a_{np_n}$$

称为 n 阶行列式，记作

$$D = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

简记作 $\Delta(a_{ij})$。数 a_{ij} 称为行列式 $\Delta(a_{ij})$ 的元素。

1.2.2 行列式计算程序实现方法分析

从定义中知道，关键是选取式(1.1)中的因子 a_{ip_i} 和计算 t 的值。将式(1.1)改写成 $(-1)^{t_1} a_{1p_1} (-1)^{t_2} a_{2p_2} \cdots (-1)^{t_n} a_{np_n}$，$t = t_1 + t_2 + \cdots + t_n$，即

$$\prod_{i=1}^{n} (-1)^{t_i} a_{ip_i} \tag{1.2}$$

其中，$t_i=\text{sign}(p_i-p_1)+\text{sign}(p_i-p_2)+\cdots+\text{sign}(p_i-p_{i-1})=\sum_{j=1}^{i-1}\text{sign}(p_i-p_j)$；

$\text{sign}(p_i-p_j)=\begin{cases}1 & p_i-p_j>0\\ 0 & p_i-p_j\leqslant 0\end{cases}$，　$i=1,2,\cdots,n$。

由此，可得行列式的计算公式为

$$\sum\prod_{i=1}^{n}(-1)^{t_i}a_{ip_i} \tag{1.3}$$

1.2.3　计算程序

Turbo C 2.0 程序

```
/*det_8.c */
#include "conio.h"
#define N 8                                    /*定义 N 为 8*/
long a[N][N];                                  /*定义元素为二维数组*/
long determinant(int n,int i,int j[ ],long d,long t,long a[ ],int
  column);
long det(int n,long a[ ],int column)           /*接口*/
{
  /*n 是行列式的阶数,a 是二维数组名,column 是行列式的最大列数*/
  /*d 是行列式,t 是项,j[i]是计算过程中选择第 i 行元素的循环变量*/
  long d,t;
  int j[N];
  d=0;
  t=1;
  d=determinant(n,0,j,d,t,a,column);
  return d;
} /*interface of computing determinant */
long determinant(int n,int i,int j[ ],long d,long t,long a[ ],int
  column)
{
  /*计算部分*/
  int k,sign,flag;
  if(i<n)
    for(j[i]=0;j[i]<n;j[i]++)
```

```
    {
      /*flag是标志*/
      flag=0;
      k=0;
      while((flag==0) && (k<i)) if(j[i]==j[k++]) flag=1;
                                          /*j[i]列已经选过*/
      if(flag==1) continue;               /*选下一列元素*/
      if(a[i*column+j[i]]==0) continue; /*元素 a[I,j[i]]为 0*/
                                          /*选下一列*/
      sign=1;for(k=0;k<i;k++) if(j[i]<j[k]) sign =-sign;
                                          /*计算符号*/
      d=determinant(n,i+1,j,d,t*sign*a[i*column+j[i]],a,column);
                                          /*递归调用*/
    }
  else d+=t;                              /*累加*/
  return d;                               /*返回行列式计算结果*/
} /*computing determinant */
main( )
{
  int i,j,n;
  long d;
  printf("n=");
  scanf("%d",&n);
  for (i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
      /*输入元素*/
      printf("a%d%d=",i+1,j+1);
      scanf("%ld",&a[i][j]);
    }
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
      printf("%ld\t",a[i][j]);printf("\n");
  }
```

```
  d=det(n,a[0],N);              /*调用 */
  printf("D=%ld\n",d);          /*行列式*/
  printf("Press any key to end.\n");
  getch( );
}
```

计算行列式

$$D=\begin{vmatrix} 3 & 1 & -1 & 2 \\ -5 & 1 & 3 & -4 \\ 2 & 0 & 1 & -1 \\ 1 & -5 & 3 & -3 \end{vmatrix}$$

```
n= 4                            /*行列式的阶数 */
a11=3
…
a44=-3
 3   1   -1   2                 /*输入的行列式元素 */
-5   1    3   -4
 2   0    1   -1
 1  -5    3   -3
D=40                            /*结果*/
```

n是行列式的阶数。D=40是运行的结果。“…”表示省略的项。

Visual C++ 6.0程序

```
#include <iostream.h>
const int N =8;                         /*定义 N 为 8 */
long a[N][N];                           /*定义元素为二维数组 */
long determinant(int n,int i,int j[ ],long d,long t,long a[ ],int
  column);
long det(int n,long a[ ],int column)   /*接口 */
{
  /*n 是行列式的阶数,a 是二维数组名,column 是行列式的最大列数 */
  long d,t;
  int j[N];
  d=0;
  t=1;
  d=determinant(n,0,j,d,t,a,column);
```

```
  return d;
} /*interface of computing determinant */
long determinant(int n,int i,int j[ ],long d,long t,long a[ ],int
  column)
{
  /*计算部分 */
  int k,sign,flag;
  if(i<n)
    for(j[i]=0;j[i]<n;j[i]++)
    {
      flag=0;
      k=0;
      while((flag==0) && (k<i)) if(j[i]==j[k++]) flag=1;
      if(flag==1) continue;
      if(a[i*column+j[i]]==0) continue;
      sign=1;
      for(k=0;k<i;k++)
      if(j[i]<j[k]) sign =-sign;
      d=determinant(n,i+1,j,d,t*sign*a[i*column+j[i]],a,column);
    }
  else d+=t;
  return d;
} /*computing determinant */
void  main( )
{
    int i,j,n;
    long d;
    cout <<"n=";cin >>n;
    for (i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
        cout <<"a"<<i+1<<j+1<<"=";cin >>a[i][j];
      }
    for(i=0;i<n;i++)
    {
```