**GaussianElimination**

在线性代数中，高斯消去法(也称为行简化法)是求解线性方程组的一种常用算法。该方法也可用于求矩阵的秩，计算矩阵的行列式，并计算可逆方阵的逆矩阵。

这一节本文描述的是最简单的高斯消去法，事实上，该方法过于简单，当主元是0是计算失败，为什么会失败，后面会给出原因。

基本行操作有三种类型，可以在矩阵的行上进行运算。对于线性方程组，运算前后的方程组等价。这些运算如下：

1. 交换两行的位置
2. 对一行元素乘以非0的常数
3. 在一行加上另一行的倍数（倍数可正可负，负数即为减去）

**消去：**关于n个未知数的n个方程有如下形式，一般称为**增广矩阵：**

![C:\Users\luk\AppData\Roaming\Tencent\Users\735343320\TIM\WinTemp\RichOle\8`XIFR8(NNVNF7L60](PDX0.png](data:image/png;base64,)

使用上面提到的三种变换方法，将上面的增广矩阵化简为上三角矩阵，其下三角部分全部转化为0元素。我们可以使用下面的代码进行消去：

Do j = 1, m - 1

If ( abs(a(j,j)) < eps ) Then

Write ( \*,'(1x,a)' ) ' The pivot is zero!'

stop

End if

Do i = j + 1, m

mult = a(i,j) / a(j,j)

Do k = j, m

a(i,k) = a(i,k) - mult \* a(j,k)

End do

b(i) = b(i) - mult \* b(j)

End do

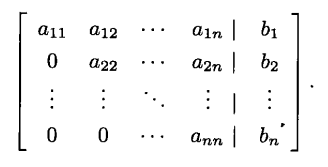
End do

前面提到，本文对与主元是0的系统会计算失败，由上述代码中：

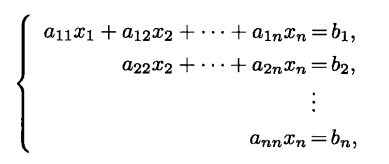
mult = a(i,j) / a(j,j)

a(j,j)表示的是主元，即对角线元素。如果其值为0，计算肯定失败。在上面的消去代码中，**m为未知数的个数**。

消去之后，最终得到：

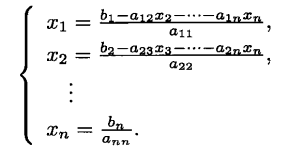


写成方程的形式：



这里需要注意的一点就是，此时的元素已经是经过变换后的元素（尽管还是写作和的形式）。

**回代：**为了完成方程最后的计算，必须进行回代过程，具体可以写为下面的式子



用代码来表示的形式如下：

Do i = m, 1, -1

Do j = i + 1, m

b(i) = b(i) - a(i,j) \* x(j)

End do

x(i) = b(i) / a(i,i)

End do

下面给出完整的代码

**Fortran代码**

!//---------------原方程-------------

!// 2x + y - z = 8

!// -3x - y + 2z = -11

!// -2x + y + 2z = -3

!//----------------------------------

!// 本代码具有一定的局限性，对主元为0或是比较小的主元，不能正确计算

!// 不过不要担心，后面会推出更新的算法

Module mod

Implicit none

Integer, parameter :: m = 3

Real(kind=8) :: a(m,m) = [ 2.d0, -3.d0, -2.d0, 1.d0, -1.d0, 1.d0, -1.d0, 2.d0, 2.d0 ]

Real(kind=8) :: b(m) = [ 8.d0, -11.d0, -3.d0 ]

Real(kind=8) :: x(m) = 0.d0

Contains

Subroutine Elimination ( ) !// 高斯消去

Implicit none

Integer :: i, j, k

Real(kind=8), parameter :: eps = 1.d-4

!// 当主元小于这个数时，程序退出

Real(kind=8) :: mult

Write ( \*,'(1x,a)' ) '经过消去前左端项与右端项为：'

Do i = 1, m

Write ( \*,'(\*(f12.5))' ) ( a(i,j), j = 1, m ), b(i)

End do

Do j = 1, m - 1

If ( abs(a(j,j)) < eps ) Then

Write ( \*,'(1x,a)' ) ' The pivot is zero!'

stop

End if

Do i = j + 1, m

mult = a(i,j) / a(j,j)

Do k = j, m

a(i,k) = a(i,k) - mult \* a(j,k)

End do

b(i) = b(i) - mult \* b(j)

End do

End do

Write ( \*,'(1x,a)' ) '经过消去后左端项与右端项为：'

Do i = 1, m

Write ( \*,'(\*(f12.5))' ) ( a(i,j), j = 1, m ), b(i)

End do

End subroutine Elimination

Subroutine BackSubstitution ( )

Implicit none

Integer :: i, j

Do i = m, 1, -1

Do j = i + 1, m

b(i) = b(i) - a(i,j) \* x(j)

End do

x(i) = b(i) / a(i,i)

End do

Write ( \*,'(1x,a)' ) '原方程解为：'

Do i = 1, m

Write ( \*,'(f12.5)' ) x(i)

End do

End subroutine BackSubstitution

End module mod

Program GaussianElimination

Use mod

Implicit none

call Elimination ( )

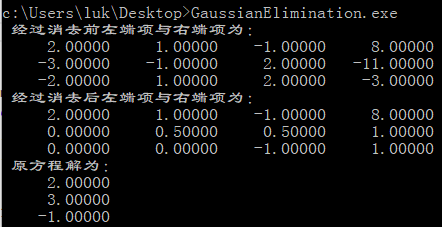
call BackSubstitution ( )

End program GaussianElimination

**Python代码**

|  |
| --- |
| """ |
|  | Created on Thu Apr 12 18:54:14 2018 |
|  |  |
|  | @author: luk |
|  | """ |
|  | #//---------------原方程------------- |
|  | #// x + y + z + w = 10 |
|  | #// 2x + 3y + z + w = 15 |
|  | #// 3x - y + 2z - w = 3 |
|  | #// 4x + y -3z + 2w = 5 |
|  | #//---------------------------------- |
|  | import numpy as np |
|  | m = 3 |
|  | eps = 1e-12 |
|  | a = np.array( [[2.0, 1.0, -1.0], [-3.0, -1.0, 2.0], [-2.0, 1.0, 2.0] ], dtype = float ) |
|  | b = np.array([ 8.0, -11.0, -3.0 ], dtype = float ) |
|  | x = np.zeros( (m,), dtype = float ) |
|  |  |
|  | print( '消去前矩阵a为：' ) |
|  | for i in range( len(a) ): |
|  | print( a[i,:] ) |
|  | print( 'please input enter:' ) |
|  | input() |
|  |  |
|  | for j in range( len(a)-1 ): |
|  | if ( np.abs(a[j,j]) < eps ): |
|  | print( ' the pivot is zero! ' ) |
|  | quit() |
|  | for i in range(j+1,len(a)): |
|  | mult = a[i,j] / a[j,j] |
|  | for k in range( j,len(a) ): |
|  | a[i,k] = a[i,k] - mult \* a[j,k] |
|  | b[i] = b[i] - mult \* b[j] |
|  |  |
|  | print( '消去后矩阵a为：' ) |
|  | for i in range( len(a) ): |
|  | print( a[i,:] ) |
|  | print( 'please input enter:' ) |
|  | input() |
|  |  |
|  | for i in range( len(a)-1,-1,-1 ): |
|  | for j in range( i, len(a) ): |
|  | b[i] = b[i] - a[i,j] \* x[j] |
|  | x[i] = b[i] / a[i,i] |
|  |  |
|  | print( '原方程解为：' ) |
|  | for i in range(len(x)): |
|  | print( x[i] ) |
|  |  |
|  | print( 'please input enter:' ) |
|  | input() |
|  |  |
|  | print( 'python求解：' ) |
|  | a = np.array( [ [1.0, 1.0, 1.0, 1.0], [2.0, 3.0, 1.0, 1.0], [3.0, -1.0, 2.0, -1.0], [4.0, 1.0, -3.0, 2.0] ], dtype = float ) |
|  | b = np.array( [ 10.0, 15.0, 3.0, 5.0 ], dtype = float ) |
|  | x = np.zeros( (4,), dtype = float ) |
|  | a = np.linalg.inv(a) # 求矩阵a的逆矩阵 |
|  | x = np.matmul(a,b) |
|  | for i in range(len(x)): |
|  | print( x[i] ) |

**计算结果如下**



最后给出代码的下载地址

Fortran代码：

<https://pan.baidu.com/s/1JDZ_Nia2ueJLWxqGCxCQbw>

Python代码：

<https://pan.baidu.com/s/1SlLb6I4R7LXjqRSMHQ4OvA>