course file

1

$A^{-1}A = I$

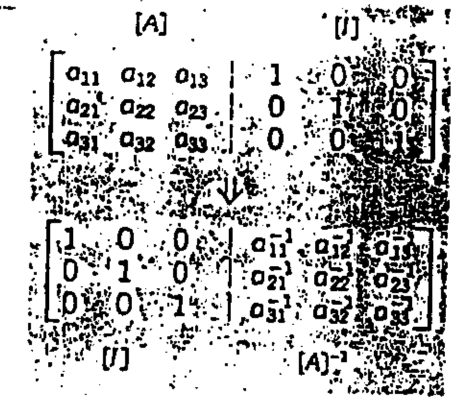

Figure 8.3
Graphical depiction of the Gauss-Jordan method, with matrix inversion. Note that the superscript −1s denote that the original values have been converted to the matrix inverse. They do not represent the value $1/a_{ij}$.

GJ TO FIND A^{-1}

EXAMPLE 8.2 Use of the Gauss-Jordan Method to Compute the Matrix Inverse

Problem Statement: Determine the matrix inverse of the system solved previou Example 7.5. Obtain the solution by multiplying $[A]^{-1}$ by the right-hand-side $\{C\}^T = [7.85 \quad -19.3 \quad 71.4]$. In addition, obtain the solution for a different righ side vector: $\{C\}^T = [20 \quad 50 \quad 15]$.

Solution: Augment the coefficient matrix with an identity matrix:

$$[A] = \left[\begin{array}{ccc|ccc} 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ 0.1 & 7 & -0.3 & 0 & 1 & 0 \\ 0.3 & -0.2 & 10 & 0 & 0 & 1 \end{array}\right]$$

Using a_{11} as the pivot element, normalize row 1 and use it to eliminate x_1 from th rows:

$$\left[\begin{array}{ccc|ccc} 1 & -0.0333333 & -0.0666667 & 0.333333 & 0 & 0 \\ 0 & 7.00333 & -0.293333 & -0.0333333 & 1 & 0 \\ 0 & -0.190000 & 10.0200 & -0.0999999 & 0 & 1 \end{array}\right]$$

Next, a_{22} can be used as the pivot element and x_2 is eliminated from the other r

$$\left[\begin{array}{ccc|ccc} 1 & 0 & -0.068057 & 0.333175 & 0.004739329 & 0 \\ 0 & 1 & -0.0417061 & -0.00473933 & 0.142180 & 0 \\ 0 & 0 & 10.0121 & -0.10090 & 0.0270142 & 1 \end{array}\right]$$

Finally, a_{33} is used as the pivot element and x_3 is eliminated from the other row

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0.332489 & 0.00492297 & 0.00679813 \\ 0 & 1 & 0 & -0.0051644 & 0.142293 & 0.00418346 \\ 0 & 0 & 1 & -0.0100779 & 0.00269816 & 0.0998801 \end{array}\right]$$

Therefore, the inverse is

From Chapra + Canale, Edition I

$$[A]^{-1} = \begin{bmatrix} 0.332489 & 0.00492297 & 0.00679813 \\ -0.0051644 & 0.142293 & 0.00418346 \\ -0.0100779 & 0.00269816 & 0.0998801 \end{bmatrix}$$

Now, the inverse can be multiplied by the first right-hand-side vector to determine the solution:

X=A⁻¹ b

$$x_1 = 7.85(0.332489) - 19.3(0.00492297) + 71.4(0.00679813)$$
$$= 3.00041181$$
$$x_2 = 7.85(-0.0051644) - 19.3(0.142293) + 71.4(0.00418346)$$
$$= -2.48809640$$
$$x_3 = 7.85(-0.0100779) - 19.3(0.00269816) + 71.4(0.0998801)$$
$$= 7.00025314$$

The second solution is simply obtained by performing another multiplication, as in

$$x_1 = 20(0.332489) + 50(0.00492297) + 15(0.00679813)$$
$$= 6.99790045$$
$$x_2 = 20(-0.0051644) + 50(0.142293) + 15(0.00418346)$$
$$= 7.0741139$$
$$x_3 = 20(-0.0100779) + 50(0.00269816) + 15(0.0998801)$$
$$= 1.43155150$$

Although the matrix inverse provides a handy and straightforward way to evaluate multiple right-hand-side vectors, it is not the most efficient algorithm for making such evaluations. In the next chapter, we will present *LU* decomposition methods that employ fewer operations to perform the same task. However, as will be elaborated in Sec. 8.1.3, the elements of the inverse are extremely useful in their own right.

Computer Algorithm for Matrix Inversion. The computer algorithm from Fig. 8.2 can be modified to calculate the matrix inverse. This involves augmenting the coefficient matrix with an identity matrix at the beginning of the program. In addition, some of the loop indices must be increased in order that the computations are performed for all the columns of the augmented coefficient matrix.

Note that an alternative method for determining the inverse will be presented Chap. 9. This approach, which is based on *LU* decomposition, will be described in 9.5.1.

INVERSE PGM FROM NAKAMURA TEXT IS ON PP 6-9 OF THIS HANDOUT

```
is version is for educational classroom use only.

To get started, type one of these: helpwin, helpdesk, or demo.
  r product information, visit www.mathworks.com.

a=[3 -.1 -.2; .1 7 -.3; .3 -.2 10]

=

    3.0000   -0.1000   -0.2000
    0.1000    7.0000   -0.3000
    0.3000   -0.2000   10.0000

inv(a)

s =

    0.3325    0.0049    0.0068
   -0.0052    0.1429    0.0042
   -0.0101    0.0027    0.0999

nv(a)*a

  =

    1.0000   -0.0000    0.0000
    0.0000    1.0000         0
   -0.0000         0    1.0000

et(a)

  =

10.3530

ank(a)

  =

    3

ref(a)

 =

    1     0     0
    0     1     0
    0     0     1
```

EXAMPLE 10-5 Matrix Multiplication

Matrix multiplication is not computed by multiplying corresponding elements of the matrices. The value in position I,J of the product of two matrices is the dot product of row I of the first matrix and column J of the second matrix, as shown in the summation equation:

$$[c(i,j)] \leftarrow \sum_{k} a(i,k) \cdot b(k,j)$$

In the equation, i and j are fixed values, and k varies in the summation.

Because dot products require that the arrays have the same number of elements, we must have the same number of elements in each row of the first matrix as we have in each column of the second matrix to compute the product of the two matrices. The product matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. Thus, if A and B both have 5 rows and 5 columns, their product has 5 rows and 5 columns. If A has 3 rows and 2 columns, and B has 2 rows and 2 columns, their product has 3 rows and 2 columns. To illustrate, matrix A is multiplied by matrix B, and the result is a new matrix C: .

$$A = \begin{bmatrix} 1.0 & 2.2 \\ 3.0 & 4.0 \\ -1.0 & 0.0 \end{bmatrix} \qquad B = \begin{bmatrix} 4.0 & -3.0 \\ 2.0 & 6.0 \end{bmatrix}$$

$$A \cdot B = C = \begin{bmatrix} 8.4 & 10.2 \\ 20.0 & 15.0 \\ -4.0 & 3.0 \end{bmatrix}$$

A subroutine to multiply two matrices must have the sizes of both input arrays in addition to the arrays themselves. The result of the multiplication must be an additional array because the original values are needed more than once in calculating the product.

Solution

In the subroutine, we print an error message if the input sizes are not correct:

```
*.......................................................*
      SUBROUTINE  MULTMX(A,AROW,ACOL,B,BROW,BCOL,
     +                   C,CROW,CCOL)
*
*  This subroutine multiplies arrays A and B
*  and stores the product in array C.
*
      INTEGER  AROW,ACOL,BROW,BCOL,CROW,CCOL,I,J,K
      REAL  A(AROW,ACOL),B(BROW,BCOL),C(CROW,CCOL)
      LOGICAL  ERROR
*
      ERROR = .FALSE.
      IF (ACOL.NE.BROW) ERROR = .TRUE.
      IF (AROW.NE.CROW) ERROR = .TRUE.
      IF (BCOL.NE.CCOL) ERROR = .TRUE.
*
```


4

```
IF (
   PF
   E
15
25
30      C(
     ENDII
*
     RETU
     END
*..........
```

In Section 10 ... ces. Systems of ... mechanical syste... current or voltag... also requires the ... from these applic... neous equations ...

10-7 APP... Eng...

In this app... equations. A gen... following form:

There are many ... choose Cramer's ... number that is c... matrix formed fr...

```
      IF (ERROR) THEN
         PRINT*, 'ERROR IN ARRAY SIZES'
      ELSE
         DO 30 I=1,CROW
            DO 25 J=1,CCOL
               C(I,J) = 0.0
               DO 15 K=1,ACOL
                  C(I,J) = C(I,J) + A(I,K)*B(K,J)
   15          CONTINUE
   25       CONTINUE
   30    CONTINUE
      ENDIF
*
      RETURN
      END
*-----------------------------------------------------------------*
```

In Section 10–7, our application solves simultaneous equations using matrices. Systems of simultaneous equations are often used in the stress analysis of mechanical systems, in the analysis of a fluid-flow system, and in the analysis of current or voltages in an electrical circuit. The design of airplane control systems also requires the solution of systems of simultaneous equations. The information from these applications is stored in a matrix, and we solve the system of simultaneous equations represented by this matrix to complete the desired analysis.

10–7 APPLICATION — DETERMINANTS (Electrical Engineering)

In this application, we develop a program to solve three simultaneous equations. A general set of three simultaneous equations can be written in the following form:

$$a(1)\,x + b(1)\,y + c(1)\,z = d(1)$$
$$a(2)\,x + b(2)\,y + c(2)\,z = d(2)$$
$$a(3)\,x + b(3)\,y + c(3)\,z = d(3)$$

There are many ways to solve a system of three simultaneous equations. We choose Cramer's Rule, which uses *determinants*. Recall that a determinant is a number that is computed from a matrix. Specifically, the determinant of the matrix formed from the set of coefficients of x, y, and z is computed as

$$\begin{vmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{vmatrix} = a1*b2*c3 + b1*c2*a3 + c1*a2*b3 - a3*b2*c1 - b3*c2*a1 - c3*a2*b1$$

APPLIED NUMERICAL METHODS IN C

SHOICHIRO NAKAMURA

PRENTICE-HALL 1993

(C) Sample Output

```
CSL/C6-1    Gauss Elimination

Augmented matrix
  0.00000e+00  -1.00000e+00   2.00000e+00   0.00000e+00
 -2.00000e+00   2.00000e+00  -1.00000e+00   0.00000e+00
 -2.00000e+00   4.00000e+00   3.00000e+00   1.00000e+00
              Machine epsilon=2.775558e-17
              Determinant = -16
 Solution
-----------------------------------------
       i        x(i)
-----------------------------------------
       1      2.187500e+00
       2      1.750000e+00
       3      1.250000e-01
-----------------------------------------
```

.1875
.2500
.125

(D) Discussions

The solution for the equation

$$\begin{pmatrix} 0 & -1 & 2 \\ -2 & 2 & -1 \\ -2 & 4 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

is found in the foregoing output. The value of the determinant indicates that the equation is well behaved. (If the determinant is extremely small, it indicates that the matrix is near singular, so the solution may not be reliable.) The machine epsilon is 2.7e-17 because the program uses double precision and is executed on VAX (see Section 1.3.4).

PROGRAM 6-2 Matrix Inversion

(A) Explanations

This program finds the inverse of a matrix A^{-1} for a nonsingular square matrix A. The inverse of a matrix can be obtained as follows: Write A and I (identity matrix) in an augmented form as $[A, I]$. Then the inverse is found where originally the identity matrix was written. Pivoting does not affect the inverse matrix computed.

In the present program, Gauss elimination is used rather than the Gauss-Jordan elimination. With Gauss elimination, each column of the identity matrix originally in the augmented matrix is viewed as a set of inhomogeneous terms. Gauss elimination for each column is done not separately but simultaneously. When the Gauss elimination is completed, the columns for the original identity matrix are filled with the solution of the Gauss elimination, which exactly comprises the inverse matrix.

Gauss elimination in this program is performed in double precision by gauss() that is essentially same as gauss() in PROGRAM 6-1.

The matrix for inversion and an identity matrix in an augmented form should be written in the declaration statement for a_init[][], which are later copied to a [][].

(B) List

```
/* CSL/c6-2.c        Matrix Inversion   */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
/*                  a[i][j] : matrix element
                    n : order of matrix
                    eps : machine epsilon                                  */
main()
{
int i, j, _i, _r;
static n = 3;
static float a_init[11][21]= {{ 2,  1,-3,   1,  0,  0},
                              {-1,  3,  2,   0,  1,  0},
                              { 3,  1,-3,   0,  0,  1}};
double a[11][21];
void gauss();
static int _aini = 1;
    {     /* Initialization of matrix elements*/
          for( j = 1; j <= 2*n; j++ )
          {   for ( i = 1 ; i <= n; i++ )  a[i][j] = a_init[i-1][j-1];
          }
     }
    printf( "\nCSL/C6-2       Matrix Inversion \n\n" );
    printf( "Original Matrix\n" );
    for( i = 1; i <= n; i++ )
    {   for( j = 1; j <= 3; j++ )  printf(  " %12.5e ", a[i][j] );
        printf(  "\n" );
    }
    gauss( n, a );
    printf( "Inverse Matrix\n" );
    for( i = 1; i <= n; i++ )
    {   printf( " " );
        for( j = n + 1; j <= (n*2); j++ )  printf( " %12.5e ", a[i][j] );
        printf( "\n" );
    }
    printf( "\n\n" );  exit(0);
}

void gauss(n, a)
int n;  double a[][21];
{
int i, j, jc, jr, k, kc, m, nv, pv;
double det, eps, eps1, eps2, r, temp, tm, va;
   eps = 1.0; eps1 = 1.0;
   while( eps1 > 0 ) {
      eps = eps/2.0; eps1 = eps*0.98 + 1.0; eps1=eps1 - 1;
   }
```

```
   eps = eps*2;
   printf( "          Machine epsilon = %11.5e \n", eps );
   eps2 = eps*2;
   det = 1.0;         /*  Initialization of determinant */
   for( i = 1; i <= (n - 1); i++ ){
      pv = i;
      for( j = i + 1; j <= n; j++ ){
         if( fabs( a[pv][i] ) < fabs( a[j][i] ) )  pv = j;
      }
      if( pv != i ){
         for ( jc = 1; jc <= (n*2); jc++ ){
            tm = a[i][jc]; a[i][jc] = a[pv][jc]; a[pv][jc] = tm;
         }
         det = -det;
      }
      if( det == 0 ){
         printf( "Matrix is singular.\n" ); exit(0);
      }
      for( jr = i + 1; jr <= n; jr++ ){
         if( a[jr][i] != 0 ){
            r = a[jr][i]/a[i][i];
            for( kc = i + 1; kc <= (n*2); kc++ ){
               temp = a[jr][kc];
               a[jr][kc] = a[jr][kc] - r*a[i][kc];
               if( fabs( a[jr][kc] ) < eps2*temp )  a[jr][kc] = 0.0;
            }
         }
      }
   }
   for( i = 1; i <= n; i++ )  det = det*a[i][i];
   printf( "          Determinant=%11.5e \n", det );
   if( a[n][n] != 0 ){
      for( m = n + 1; m <= (n*2); m++ ){
         a[n][m] = a[n][m]/a[n][n];
         for( nv = n - 1; nv >= 1; nv-- ){
            va = a[nv][m];
            for(k=nv+1; k<=n; k++) va=va - a[nv][k]*a[k][m];
            a[nv][m] = va/a[nv][nv];
         }
      }  return;
   }
}
```

(C) Sample Output

```
CSL/C6-2      Matrix Inversion

Original Matrix
  2.00000e+00   1.00000e+00  -3.00000e+00
 -1.00000e+00   3.00000e+00   2.00000e+00
  3.00000e+00   1.00000e+00  -3.00000e+00
          Machine epsilon = 2.77556e-17
          Determinant=1.10000e+01
Inverse Matrix
 -1.00000e+00   0.00000e+00   1.00000e+00
  2.72727e-01   2.72727e-01  -9.09091e-02
 -9.09091e-01   9.09091e-02   6.36364e-01
```