Journal of Chemical, Biological and Physical Sciences

An International Peer Review E-3 Journal of Sciences

Available online at www.jcbsc.org

Section C: Physical Science

CODEN (USA): JCBPAT

Research Article

Comparison of Numerical Efficiencies of Gaussian Elimination and Gauss-Jordan Elimination Methods in Finding the Inverses of Matrices

R. B. Srivastava*and Vinod Kumar

Department of Mathematics Department, M. L. K. P. G. College, Balrampur, U. P., India

Received: 18 October 2012; **Revised** 30 October 2012; **Accepted:** 5 November 2012

Abstract: **Inverses of matrices have been found out using Gaussian and Gauss-Jordan elimination methods with the help of developed computer program. Numbers of operations involved in the determination of inverses of matrices have also been computed. Numerical efficiencies of Gaussian and Gauss-Jordan elimination methods have been found to be 2.547511 and 2.392344 which indicates that the Gaussian elimination method is approximately 1.051 times faster than the Gauss-Jordan elimination method**

Keywords: Numerical efficiency, Gaussian elimination method, Gauss-Jordan elimination method, numerical efficiency, matrix inversion, identity matrix.

INTRODUCTION

One way how to solve Ax = b consists in computation of A^{-1} complemented by its multiplication by the right hand side vector b. Such a process, however, is highly inefficient, since the inverse matrix computation generally requires n^3 operations, where n the matrix order. The consequent matrix-vector multiplication adds additional n^2 operations. The solution of the system of algebraic equations with one right hand side requires only $n^3/3+n^2-n/3$ operations [1-5].

There is another, even more significant reason against matrix inversion, namely the loss of symmetry and banddedness of the matrix being inverted. This prevents usage of any efficient matrix storage modes.[6-9] A few decades ago solving a system of equations by the matrix inversion was considered to be a sort of computational crime. Today, when a 1000 × 1000 matrix can be inverted in a fraction of second executing the simple Matlab statementinv(A), one can proceed unpunished for doing this. But, as before, it is the problem size and its 'standardness' which influences the decision of an acceptable brute force solvability.

The limits are higher than before, as well as the consequences and fines we have pay for our eventual failures. So if we insist on the computing the matrix inversion we still could do in a memory-efficient way by solving the system of equations n times[10-13]

$$Ax^{(i)}=e^{(i)},\ i=1,2,\cdots n,$$

with right hand side vectors $e^{(i)}$ containing zeros with the exception of a single '1' at its i-th location. Each solution gives a single vector $x^{(i)}$ which is just the i-th column of the inverse matrix A^{-1} , i.e.

$$A^{-1}=[x^{(1)}x^{(2)}\cdots x^{(n)}].$$

This way we can carry out the triangularization of the matrix only once employing thus the suitable efficient matrix storage schemes. Obtaining the inverse this way requires the $4n^3/3$ operations, which is still more than using classical Gauss-Jordan reduction process. It might be acceptable if the memory limitations are more severe than those of time efficiency.[14-17]

For inverting a matrix, *Gauss-Jordan elimination* is about as efficient as any other method. For solving sets of linear equations, Gauss-Jordan elimination produces *both* the solution of the equations for one or more right-hand side vectors **b**, and also the matrix inverse $\mathbf{A}^{-1}$. However, its principal weaknesses are (i) that it requires all the right-hand sides to be stored and manipulated at the same time, and (ii) that when the inverse matrix is *not* desired, Gauss-Jordan is three times slower than the best alternative technique for solving a single linear set. The method's principal strength is that it is as stable as any other direct method, perhaps even a bit more stable when full pivoting is used.[18-22]

MATERIAL AND METHOD

In order to find out the inverse of Matrix A, we take identity matrix B of same order. Now, we apply same elementary operations on A and B to reduce A into identity matrix. When A is reduced into identity matrix, the matrix B becomes the inverse of A. In elementary operations, we do the following-[10-13]

- Interchanging any two *rows* of A and the corresponding *rows* of the B.
- Replacing any row in A by a linear combination of itself and any other row. Same operation has to be applied on B.
- Interchanging any two *columns* of A**.** Same operation has to be applied on B.

Inverses of matrices given in Table-1 have been found out using Gaussian and Gauss-Jordan elimination methods with the help of C++ computer programs given in program-1 and program-2. Number of operations involved in finding out the inverse of each matrix has also been calculated.

RESULT AND DISCUSSION

Inverse of Matrix-1 by Gaussian Elimination Method-

Matrices A and B are as under-

Matrix A is given below

1.000	2.000	3.000
2.000	4.000	-1.000
-3.000	1.000	5.000

Matrix B is given below

1.000	0.000	0.000
0.000	1.000	0.000

| 0.000 0.000 1.000 |

Multiplying first row by 2.000 and subtracting it from 2 row

Matrix A is given below

| 1.000 2.000 3.000 |

| 0.000 0.000 -7.000 |

| -3.000 1.000 5.000 |

Matrix B is given below

| 1.000 0.000 0.000 |

| -2.000 1.000 0.000 |

| 0.000 0.000 1.000 |

Multiplying first row by -3.000 and subtracting it from 3 row

Matrix A is given below

| 1.000 2.000 3.000 |

| 0.000 0.000 -7.000 |

| 0.000 7.000 14.000 |

Matrix B is given below

| 1.000 0.000 0.000 |

| -2.000 1.000 0.000 |

| 3.000 0.000 1.000 |

Swapping 2 and 3 rows

Matrix A is given below

| 1.000 2.000 3.000 |

| 0.000 7.000 14.000 |

| 0.000 0.000 -7.000 |

Matrix B is given below

| 1.000 0.000 0.000 |

| 3.000 0.000 1.000 |

| -2.000 1.000 0.000 |

Matrix A is given below

| 1.000 -0.000 -0.000 |

| 0.000 7.000 0.000 |

| 0.000 0.000 -7.000 |

Matrix B is given below

| 0.429 -0.143 0.000 |

| -1.000 2.000 1.000 |

| -2.000 1.000 0.000 |

The inverse of matrix is

| 0.429 -0.143 0.000 |

| -0.143 0.286 0.143 |

| 0.286 -0.143 -0.000 |

Number of operations performed in finding out the inverse of Matrix-1 by Gaussian Elimination Method = 64

Calculation of inverse of Matrix-1 by Gauss-Jordan Elimination method

Matrices A and B are as under-

Matrix A is given below

| 1.000 2.000 3.000 |

| 2.000 4.000 -1.000 |

| -3.000 1.000 5.000 |

Matrix B is given below

| 1.000 0.000 0.000 |

| 0.000 1.000 0.000 |

| 0.000 0.000 1.000 |

The inverse of matrix is

| 0.429 -0.143 -0.286 |

| -0.143 0.286 0.143 |

| 0.286 -0.143 -0.000 |

Total number of calculations in finding out the inverse of Matrix-1 by Gauss-Jordan Elimination method = 66

Program-1: To find Inverses of matrices using Gaussian elimination method

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
//Gauss elimination method
float a[15][15],x[15],b[15][15];
int n;
int c=0;
FILE *fpt;
void main(void)
```

```
{
  float r,s;
  int i,j,k,l,m,p;
  void swap(int r1, int r2);
  void displaymat(int row);
  fpt=fopen("vincinvg.doc", "a");
  clrscr();
  printf("Number of rows in square matrix=");
  scanf("%d",&n);
  printf("Give the matrix row-wise\n");
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
         printf("a[%2d,%2d]=", i,j);
         scanf("%f",&a[i][j]);
   }
  }
 // Adding identity matrix
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
     if (i==j) b[i][j]=1; else b[i][j]=0;
   }
  }
  printf("Matrix is as follows-\n\n");
  fprintf(fpt,"Matrix is as follows-\n\n");
  displaymat(n);
  //Matrix input finished
  //Calculating inverse
  for(i=1;i<n;i++)
  {
```

```
if (a[i][i]==0)
{
    for(j=i+1; j<=n; j++)
    {
     if (a[j][i]!=0)
     {
        swap(i,j);
        goto afterswap;
     }
     else
      {
        if (j==n)
        {
            printf("Matrix is singular as a[%2d,%2d]=0\n",j,i);
            fprintf(fpt,"Matrix is singular as a[%2d,%2d]=0\n",j,i);
            goto last;
        }
      }
     }
  }
afterswap: ;
  for(k=i+1;k<=n;k++)
  {
    if (a[i][i]==0)
    {
        printf("Matrix singular with i=%2d\n",i);
        fprintf(fpt,"Matrix singular with i=%2d\n",i);
        displaymat(n);
        goto last;
    }
    r=a[k][i]/a[i][i];
    c++;
    for(l=1;l<=n;l++)
```

```
        {
                a[k][l]=a[k][l]-r*a[i][l];
                b[k][l]=b[k][l]-r*b[i][l];
                c=c+2;
        }
        if (r != 0)
        {
                 printf("Multiplying first row by %7.3f and subtracting it from %d row\n\n",r,k);
                 fprintf(fpt,"Multiplying first row by %7.3f and subtracting it from %d row\n\n",r,k);
        }
     }
  }
  //reducing upper triangular to indentity matrix
for(k=n;k>1;k--)
{
   for(i=1;i<k;i++)
   {
            r=a[i][k]/a[k][k];
            c++;
            for(j=1;j<=k;j++)
            {
              a[i][j]=a[i][j]-r*a[k][j];
              c++;
              b[i][j]=b[i][j]-r*b[k][j];
              c++;
            }
     }
  }
    for(i=1;i<=n;i++)
   {
     if (a[i][i] != 1)
     {
            for(p=1;p<=n;p++)
```

```
            {
              b[i][p]=b[i][p]/a[i][i];
              c++;
            }
            a[i][i]=1;
    }
}
printf("The inverse of matrix is \n");
fprintf(fpt,"The inverse of matrix is \n");
 for(i=1; i<=n; i++)
 {
        printf("│ ");
        fprintf(fpt,"│ ");
        for(j=1;j<=n;j++)
        {
           printf("%9.3f ",b[i][j]);
           fprintf(fpt,"%9.3f ",b[i][j]);
        }
        fprintf(fpt,"│ ");
        printf("│ ");
        printf("\n");
        fprintf(fpt,"\n");
 }
        printf("\n ");
        printf("\n");
        fprintf(fpt,"\n");
        fprintf(fpt,"\n");
last: ;
printf("Total number of calculations = %5d\n",c);
fprintf(fpt,"Total number of calculations = %5d\n",c);
printf("Program finished\n");
printf("Press any key to terminate\n");
getch();
```

```
}
  void swap(int r1, int r2)
  {
   float t;
   int i;
   for(i=1;i<=n;i++)
   {
        t=a[r1][i];
        a[r1][i]=a[r2][i];
        a[r2][i]=t;
        t=b[r1][i];
        b[r1][i]=b[r2][i];
        b[r2][i]=t;
        c=c+6;
   }
  }
void displaymat(int row)
{
   int p, q, col=row;
   printf("Matrix A is given below\n");
   fprintf(fpt,"Matrix A is given below\n");

   for(p=1; p<=row; p++)
   {
        printf(" | ");
        fprintf(fpt," | ");
        for(q=1;q<=col;q++)
        {
          printf("%9.3f ",a[p][q]);
          fprintf(fpt,"%9.3f ",a[p][q]);
        }
        printf(" | ");
        fprintf(fpt," | ");
```

```
            printf("\n");
            fprintf(fpt,"\n");
        }
        printf("\n");
        fprintf(fpt,"\n");
        printf("\n");
        fprintf(fpt,"\n");
        printf("Matrix B is given below\n");
        fprintf(fpt,"Matrix B is given below\n");
        for(p=1; p<=row; p++)
        {
            printf("│ ");
            fprintf(fpt,"│ ");
            for(q=1;q<=col;q++)
            {
               printf("%9.3f ",b[p][q]);
               fprintf(fpt,"%9.3f ",b[p][q]);
            }
            printf("│ ");
            fprintf(fpt,"│ ");
            printf("\n");
            fprintf(fpt,"\n");
        }
        printf("\n");
        fprintf(fpt,"\n");
        printf("\n");
        fprintf(fpt,"\n");
        //getch();
}
```

Program-2: To find Inverses of matrices using Gauss-Jordan elimination method

```
#include<conio.h>
```

```
#include<stdio.h>
#include<math.h>
//Matrix inverse by Gauss Jordan elimination method
float a[15][15],x[15],b[15][15];
int n;
FILE *fpt;
void main(void)
{
  float r,s;
  int i,j,k,l,m,p;
  void swap(int r1, int r2);
  void displaymat(int row);
  fpt=fopen("vininvgj.doc", "a");
  clrscr();
  printf("Number of rows in square matrix=");
  scanf("%d",&n);
  printf("Give the matrix row-wise\n");
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
        printf("a[%2d,%2d]=", i,j);
        scanf("%f",&a[i][j]);
   }
  }
 // Adding identity matrix
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
    if (i==j) b[i][j]=1; else b[i][j]=0;
   }
  }
```

```
printf("Matrix is as follows-\n\n");
fprintf(fpt,"Matrix is as follows-\n\n");
displaymat(n);
//Matrix input finished
//Calculating inverse
 for(j=1;j<=n;j++)
 {
   if (a[j][j]==0)
   {
         if (j<n) swap(j,j+1);
         if (j==n)
         {
           printf("Singular matrix\n");
           fprintf(fpt,"Singular matrix\n");
           goto last;
         }
   }
   for(i=1;i<=n;i++)
   {
        if (i != j)
        {
          s=a[i][j]/a[j][j];
          for(k=1;k<=n;k++)
          {
            a[i][k] = a[i][k] - a[j][k] * s;
            b[i][k] = b[i][k] - b[j][k] * s;
          }
            displaymat(n);
           //  getch();
        }
   }
 }
for(i=1;i<=n;i++)
```

```
{
  if (a[i][i] != 1)
  {
        for(p=1;p<=n;p++)
          b[i][p]=b[i][p]/a[i][i];
        a[i][i]=1;
  }
}
displaymat(n);
printf("The inverse of matrix is \n");
fprintf(fpt,"The inverse of matrix is \n");
 for(i=1; i<=n; i++)
 {
        printf("│ ");
        fprintf(fpt,"│ ");
        for(j=1;j<=n;j++)
        {
           printf("%9.3f ",b[i][j]);
           fprintf(fpt,"%9.3f ",b[i][j]);
        }
        fprintf(fpt,"│ ");
        printf("│ ");
        printf("\n");
        fprintf(fpt,"\n");
 }
        printf("\n ");
        printf("\n");
        fprintf(fpt,"\n");
        fprintf(fpt,"\n");
last: ;
printf("Program finished\n");
printf("Press any key to terminate\n");
getch();
```

```
}
  void swap(int r1, int r2)
  {
   float t;
   int i;
   for(i=1;i<=n;i++)
   {
        t=a[r1][i];
        a[r1][i]=a[r2][i];
        a[r2][i]=t;
        t=b[r1][i];
        b[r1][i]=b[r2][i];
        b[r2][i]=t;
   }
  }
void displaymat(int row)
{
   int p, q, col=row;
   printf("Matrix A is given below\n");
   fprintf(fpt,"Matrix A is given below\n");
   for(p=1; p<=row; p++)
   {
        printf(" | ");
        fprintf(fpt," | ");
        for(q=1;q<=col;q++)
        {
          printf("%9.3f ",a[p][q]);
          fprintf(fpt,"%9.3f ",a[p][q]);
        }
        printf(" | ");
        fprintf(fpt," | ");
        printf("\n");
        fprintf(fpt,"\n");
```

```
    }
    printf("\n");
    fprintf(fpt,"\n");
    printf("\n");
    fprintf(fpt,"\n");
    printf("Matrix B is given below\n");
    fprintf(fpt,"Matrix B is given below\n");
    for(p=1; p<=row; p++)
    {
        printf("│ ");
        fprintf(fpt,"│ ");
        for(q=1;q<=col;q++)
        {
          printf("%9.3f ",b[p][q]);
          fprintf(fpt,"%9.3f ",b[p][q]);
        }
        printf("│ ");
        fprintf(fpt,"│ ");
        printf("\n");
        fprintf(fpt,"\n");
    }
    printf("\n");
    fprintf(fpt,"\n");
    printf("\n");
    fprintf(fpt,"\n");
    //getch();
}
```

Table-1: Matrices whose inverses have been found out by Gaussian and Gauss-Jordan elimination methods

S. No.	Matrix
1	1 2 3 2 4 –1 –3 1 5
2	1 2 3 –3 1 5 2 4 –1
3	1 2 3 2 4 –1 3 1 5
4	4 8 4 0 1 4 7 2 1 5 4 –3 1 3 0 –2
5	4 3 4 0 1 2 3 2 1 5 4 –3 1 0 4 7
6	2 1 –1 –1 –3 1 1 8 –2
7	9 3 1 0 5 2 3 2 7 5 8 3 1 2 4 7
8	–9 2 1 0 2 –2 3 2 1 7 –8 3 –1 2 4 –7
9	3 1 1 0 2 2 3 3 1 4 6 3 –1 1 8 –7
10	3 1 4 1 2 2 8 3 1 1 –2 3 2 1 8 –7

S. No.	Matrix
11	1 2 3 4 3 2 1 4 6 7 8 9 2 7 6 5
12	2 2 3 4 1 2 1 4 3 7 3 9 7 7 2 5
13	0 2 3 4 1 2 0 4 3 0 3 8 1 2 2 5
14	0 2 3 1 2 2 5 4 1 0 2 8 3 2 2 5
15	1 2 4 1 2 1 2 4 1 9 0 2 5 1 1 3
16	1 2 4 5 5 1 1 3 1 5 0 2 2 7 3 1
17	1 2 0 5 0 1 3 2 2 3 5 2 3 0 2 1
18	1 3 1 4 2 1 3 2 1 3 4 2 3 7 2 1
19	4 2 1 5 1 7 3 2 2 5 2 1 3 0 2 4

S. No.	Matrix
20	3 1 1 4 2 6 1 1 3 4 3 2 4 0 1 3

Table-2: Number of calculations in the determination of inverse of matrix by Gaussian and Gauss-Jordan elimination methods

Matrix	Number of calculations in the determination of inverses of matrices by Gaussian elimination	Number of calculations in the determination of inverses of matrices by Gauss-Jordan elimination
1	64	66
2	46	48
3	64	66
4	116	124
5	116	124
6	Singular	Singular
7	116	124
8	116	124
9	116	124
10	116	124
11	136	144
12	112	120
13	136	144
14	136	144
15	112	120
16	112	120
17	108	116
18	112	120
19	116	124
20	116	124
Total no. of calculations	2066	2200

CONCLUSION

Efficiency of matrix inversion by depends on the number of calculations involved in the process of determination of inverse of matrix. As the number of these calculations increases; the efficiency of the method decreases and vice-versa. Number of calculations involved in the determination of inverse of matrix by Gaussian and Gauss-Jordan elimination methods have also been found out and these are given in Table-2. Let us define the efficiency with the help of formula-

Efficiency of method for matrix inversion (η) = 100000/(sum of number of calculations in the determination of inverse of matrix x number of matrices)

In order to find out the efficiency of Gaussian elimination method, one should take many matrices; but here only 20 matrices have been considered in which one is singular.

Efficiency of Gaussian elimination method (η) = 100000/(sum of number of calculations in the determination of inverse of matrix by Gaussian elimination method x number of matrices)

= 100000/(2066 x 19)

= 2.547511

Efficiency of Gauss-Jordan elimination method in the determination of inverses of matrices (η)

= 100000/(sum of number of calculations in determination of inverse of matrix x number matrices)

= 100000/(2200 x 19)

= 2.392344

Speed of Gaussian elimination method for matrix inversion over Gauss-Jordan elimination method (S) is defined as-

S = efficiency of Gaussian elimination method / efficiency of Gauss-Jordan elimination method

= 2.547511/ 2.392344

= 1.051

Thus, Gaussian elimination method is approximately 1.051 times faster than Gauss-Jordan elimination method.

REFERENCES

1. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM, Philadelphia, PA, 1994.
2. K. J. Bathe.Numerical methods in finite element analysis. Prentice-Hall, Inc., Englewood Cliffs, 1976.
3. K. J. Bathe and E. L. Wilson. Numerical methods in finite element analysis. Prentice-Hall, Englewood Cliffs, 1976.
4. G. Cantin. An equation solver of very large capacity. International Journal for Numerical Methods in Engineering, 3:379–388, 1971.
5. G. E. Forsythe, M.A. Malcolm, and C. B. Moler. Computer methods for mathematical computations. Prentice-Hall, Englewood Cliffs, 1977.

6. G. H. Golub and Ch. F VanLoan. Matrix computation. John Hopkins, ISBN 978-0-8018-5414-9, New York, 1996.
7. L.A. Hageman and D.M. Young. Applied Iterative Methods. Academic, in Russian,Moskva Mir, 1986.142
8. Owen D.R.J. Hinton, E.Finite Element Programming. Academic Press, London, 1977.
9. R.A. Horne and C.A. Johnson. Matrix analysis. Cambridge University Press, 1993.
10. B.M. Irons. A frontal solution program for finite elemenent analysis. International Journal for Numerical Methods in Engineering, 2:5–32, 1970.
11. C. Moler, J. Little, and S. Bangert. PC-Matlab. The MathWorks, Inc., Sherborn, MA, 1987.
12. S. Oliviera and D. Steward. Writing scientific software. Cambridge University Press, New York, 2006.
13. Y. C. Pao. Algorithms for direct-access Gaussian solution of structural stiffness matrix equation. International Journal for Numerical Methods in Engineering, 12:751–764, 1978.
14. M. Papadrakakis. Solving Large-Scale Problems in Mechanics. John Wiley and SonsLtd, Chichester, Baffins Lane, 1993.
15. B. N. Parlett. The Symmetric Eigenvalue Problem. Prentice-Hall, Englewood Cliffs, N. J., 1978.
16. S. R. Phansalkar.Matrix Iterative Methods for Structural Reanalysis, Vol. 4. Com-puters & Structures, 1974.
17. J.S. Przemiecki. Theory of matrix structural analysis. McGraw-Hill, New York,1968.
18. A. Ralston.A first course in numerical analysis. McGraw-Hill, New York, 1965.
19. J. R. Rice. Matrix Computation and mathematical software. McCraw-Hill, Auckland,1983.
20. G. Strang.Linear algebra and its applications. Academic Press, Englewood Cliffs,1976.
21. R. S. Varga. Matrix Iterative Analysis. Prentice-Hall, New Jersey, 1965.
22. J. H. Wilkinson. Modern error analysis. SIAM Review, 13:751–764, 1971

*__Correspondence Author: R. B. Srivastava;__ Associate Prof., Mathematics Department
M. L. K. P. G. College, Balrampur, U. P., India; Email: rambux@gmail.com