Scientific Research Publishing

Applied Mathematics, 2018, 9, 240-249
http://www.scirp.org/journal/am
ISSN Online: 2152-7393
ISSN Print: 2152-7385

Visual Basic™ Routine for In-Place Matrix Inversion

Debabrata DasGupta[1,2,3]

[1]Former V.P.-Development, LEAP Software, Inc., Tampa, FL, USA
[2]Former Principal Consultant, McDonnell Douglas Automation Co., St. Louis, MO, USA
[3]Former Assistant Director, Central Water & Power Commission, New Delhi, India
Email: ddasgupta@email.com

How to cite this paper: DasGupta, D. (2018) Visual Basic™ Routine for In-Place Matrix Inversion. *Applied Mathematics*, **9**, 240-249.
https://doi.org/10.4236/am.2018.93018

Received: January 15, 2018
Accepted: March 26, 2018
Published: March 29, 2018

Abstract

A modified version of the Gauss-Jordan algorithm for performing In-Place matrix inversion without using an augmenting unit matrix was described in a previous article by the author. He had also developed several Structural Engineering softwares during his career using that method as their analysis engine. He chose matrix inversion because it was suitable for in-core solution of large numbers of vectors for the same set of equations as encountered in structural analysis of moving, dynamic and seismic loadings. The purpose of this article is to provide its readers with its theoretical background and detailed computations of an In-Place matrix inversion task as well as a Visual Basic routine of the algorithm for direct incorporation into Visual Basic 6™ softwares and Visual Basic for Applications™ macros in MS-Excel™ spreadsheets to save them time and effort of software development.

Keywords

VB6, VBA, FORTRAN, MS-Excel, Numerical Methods, Gauss-Jordan, Matrix Methods, Matrix Inversion, In-Place Inversion, Structural Analysis

1. Introduction

Gauss-Jordan is a standard matrix inversion procedure developed in 1887 [1]. It requires the original matrix to be appended by a unit (identity) matrix and after the inversion operation is completed the original matrix is transformed into a unit matrix while the appended unit matrix becomes the inverse.

A detailed description of the original Gauss-Jordan method as well as its comparison with the author's In-Place version which does not require the aug-

menting matrix and the reason for its development were previously published by him in another article [2].

This paper reproduces relevant parts of that article appended by the computational details of an example and a Visual Basic routine.

2. Mathematics

The author has used certain terms in the following discussion which are defined as follows: *Normalization* is dividing an entire row by its pivotal element to transform the pivotal element to unity; *Virtualization* is replicating an element or a vector of the current augmenting matrix within the original matrix space without creating the real unit matrix; the *Complementary* of a component of the original matrix is its corresponding component in the virtual augmenting matrix and the *Reduction* of a row is the modification of the pivotal row by the ratio of the row element on the pivotal column and the pivotal element and then subtracting it from the row, thereby reducing the row element on the pivotal column to zero.

Unlike the classical Gauss-Jordan method, the author's ***In-Place Inversion*** algorithm does not require augmenting with and performing operations on an identity matrix and the procedure is described below:

Just as with Gauss-Jordan, the following two operations are iterated on all rows to obtain the inverse.

Operation 1: The unpivoted row with the largest absolute diagonal element is selected as the pivotal row p and the value of its pivotal element $A_{p,p}$ is saved as the pivot P_p after which the pivotal element $A_{p,p}$ is replaced by unity (1) to virtualize the complementary element $A_{p,p+n}$ of the unit matrix. Then the pivotal row p is normalized by dividing the entire row by P_p, *i.e.*, $A_{p,j} = A_{p,j} / P_p$ where $j = 1 \to n$. This changes the pivotal element $A_{p,p}$ to $1/P_p$ which replicates the current value of its complementary element within the virtual unit matrix.

Operation 2: Each non-pivotal row i, *i.e.*, $i = 1 \to n \neq p$, is reduced by saving the value of its pivotal column element $A_{i,p}$ as P_i, recomputing all elements in the current row i as $A_{i,j} = A_{i,j} - A_{p,j} * P_i$ where $j = 1 \to n$ and then resetting $A_{i,p}$ to 0 to minimize truncation error. As mentioned earlier, this is a shorter version of the operation $A_{i,j} = A_{i,j} - A_{p,j} * P_i / A_{p,p}$ where $j = 1 \to n$ since $A_{p,p}$ = 1 in the original matrix after normalization of the pivotal row.

This procedure implicitly duplicates the functionality of the unit matrix of the Gauss-Jordan method within the original matrix. After performing these two operations on every row, treating each row once as a pivotal row, the original matrix is replaced by its inverse.

The sequence of operations 1 and 2 can be reversed but in that case the pivotal element $A_{p,p}$ will not be unity during operation 2 and the explicit formula will have to be used, thereby substantially increasing the amount of

computation.

Solutions of the same example by both Gauss-Jordan and In-Place algorithms are given in the Author's original article for comparison purposes and hence are not reproduced here.

Computational details of a 3×3 matrix [3] as well as a Visual Basic subroutine with a sample calling routine are given below. It is equally useful for VB6 programs as well as a VBA routine in MS-Excel™ spreadsheets. Conversion of the routine into other languages is also quite simple.

3. Computational Details of In-Place Inversion of a 3 × 3 Matrix [3]

```
Initialization
==============

  Preset all rows to Active

  Current Matrix:
      -1.0000     -1.0000      3.0000  Active
       2.0000      1.0000      2.0000  Active
      -2.0000     -2.0000      1.0000  Active

Cycle 1
=======

  Currently active row with largest (absolute) diagonal = 1

  Process Pivotal Row 1
  Save element (1,1) = -1.0000 as Pivot
  Reset Element (1,1) = 1.0000 to 1.
  Reset Pivotal Row 1 to Inactive

  Current Matrix:
       1.0000     -1.0000      3.0000  Inactive
       2.0000      1.0000      2.0000  Active
      -2.0000     -2.0000      1.0000  Active
  Divide all elements of pivotal row 1 by Pivot = -1.0000

  Current Matrix:
      -1.0000      1.0000     -3.0000  Inactive
       2.0000      1.0000      2.0000  Active
      -2.0000     -2.0000      1.0000  Active

  Process non-pivotal row 2
  Retrieve Pivotal column element (2,1) = 2.0000 and reset it to 0.

  Current Matrix:
      -1.0000      1.0000     -3.0000  Inactive
       0.0000      1.0000      2.0000  Active
      -2.0000     -2.0000      1.0000  Active

  Subtract from each element in current row 2.0000 * corresponding element in pivotal row 1

  Previous element (2,1) = 0.0000 is now 0.0000 - 2.0000 * -1.0000 = 2.0000
  Previous element (2,2) = 1.0000 is now 1.0000 - 2.0000 * 1.0000 = -1.0000
  Previous element (2,3) = 2.0000 is now 2.0000 - 2.0000 * -3.0000 = 8.0000

  Current Matrix:
      -1.0000      1.0000     -3.0000  Inactive
       2.0000     -1.0000      8.0000  Active
      -2.0000     -2.0000      1.0000  Active

  Process non-pivotal row 3
  Retrieve Pivotal column element (3,1) = -2.0000 and reset it to 0.

  Current Matrix:
      -1.0000      1.0000     -3.0000  Inactive
       2.0000     -1.0000      8.0000  Active
       0.0000     -2.0000      1.0000  Active

  Subtract from each element in current row -2.0000 * corresponding element in pivotal row 1

  Previous element (3,1) = 0.0000 is now 0.0000 - -2.0000 * -1.0000 = -2.0000
  Previous element (3,2) = -2.0000 is now -2.0000 - -2.0000 * 1.0000 = 0.0000
  Previous element (3,3) = 1.0000 is now 1.0000 - -2.0000 * -3.0000 = -5.0000

  Current Matrix:
      -1.0000      1.0000     -3.0000  Inactive
       2.0000     -1.0000      8.0000  Active
      -2.0000      0.0000     -5.0000  Active
```

```
Cycle 2
=======

  Currently active row with largest (absolute) diagonal = 3

  Process Pivotal Row 3
  Save element (3,3) = -5.0000 as Pivot
  Reset Element (3,3) = 1.0000 to 1.
  Reset Pivotal Row 3 to Inactive

  Current Matrix:
      -1.0000     1.0000    -3.0000  Inactive
       2.0000    -1.0000     8.0000  Active
      -2.0000     0.0000     1.0000  Inactive
  Divide all elements of pivotal row 3 by Pivot = -5.0000

  Current Matrix:
      -1.0000     1.0000    -3.0000  Inactive
       2.0000    -1.0000     8.0000  Active
       0.4000     0.0000    -0.2000  Inactive

  Process non-pivotal row 1
  Retrieve Pivotal column element (1,3) = -3.0000 and reset it to 0.

  Current Matrix:
      -1.0000     1.0000     0.0000  Inactive
       2.0000    -1.0000     8.0000  Active
       0.4000     0.0000    -0.2000  Inactive

  Subtract from each element in current row -3.0000 * corresponding element in pivotal row 3

  Previous element (1,1) = -1.0000 is now -1.0000 - -3.0000 * 0.4000 = 0.2000
  Previous element (1,2) = 1.0000 is now 1.0000 - -3.0000 * 0.0000 = 1.0000
  Previous element (1,3) = 0.0000 is now 0.0000 - -3.0000 * -0.2000 = -0.6000

  Current Matrix:
       0.2000     1.0000    -0.6000  Inactive
       2.0000    -1.0000     8.0000  Active
       0.4000     0.0000    -0.2000  Inactive

  Process non-pivotal row 2
  Retrieve Pivotal column element (2,3) = 8.0000 and reset it to 0.

  Current Matrix:
       0.2000     1.0000    -0.6000  Inactive
       2.0000    -1.0000     0.0000  Active
       0.4000     0.0000    -0.2000  Inactive

  Subtract from each element in current row 8.0000 * corresponding element in pivotal row 3

  Previous element (2,1) = 2.0000 is now 2.0000 - 8.0000 * 0.4000 = -1.2000
  Previous element (2,2) = -1.0000 is now -1.0000 - 8.0000 * 0.0000 = -1.0000
  Previous element (2,3) = 0.0000 is now 0.0000 - 8.0000 * -0.2000 = 1.6000

  Current Matrix:
       0.2000     1.0000    -0.6000  Inactive
      -1.2000    -1.0000     1.6000  Active
       0.4000     0.0000    -0.2000  Inactive


Cycle 3
=======

  Currently active row with largest (absolute) diagonal = 2

  Process Pivotal Row 2
  Save element (2,2) = -1.0000 as Pivot
  Reset Element (2,2) = 1.0000 to 1.
  Reset Pivotal Row 2 to Inactive

  Current Matrix:
       0.2000     1.0000    -0.6000  Inactive
```

```
    -1.2000     1.0000     1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive
 Divide all elements of pivotal row 2 by Pivot = -1.0000

 Current Matrix:
     0.2000     1.0000    -0.6000  Inactive
     1.2000    -1.0000    -1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive

 Process non-pivotal row 1
 Retrieve Pivotal column element (1,2) = 1.0000 and reset it to 0.

 Current Matrix:
     0.2000     0.0000    -0.6000  Inactive
     1.2000    -1.0000    -1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive

 Subtract from each element in current row 1.0000 * corresponding element in pivotal row 2

 Previous element (1,1) = 0.2000 is now 0.2000 - 1.0000 * 1.2000 = -1.0000
 Previous element (1,2) = 0.0000 is now 0.0000 - 1.0000 * -1.0000 = 1.0000
 Previous element (1,3) = -0.6000 is now -0.6000 - 1.0000 * -1.6000 = 1.0000

 Current Matrix:
    -1.0000     1.0000     1.0000  Inactive
     1.2000    -1.0000    -1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive

 Process non-pivotal row 3
 Retrieve Pivotal column element (3,2) = 0.0000 and reset it to 0.

 Current Matrix:
    -1.0000     1.0000     1.0000  Inactive
     1.2000    -1.0000    -1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive

 Subtract from each element in current row 0.0000 * corresponding element in pivotal row 2

 Previous element (3,1) = 0.4000 is now 0.4000 - 0.0000 * 1.2000 = 0.4000
 Previous element (3,2) = 0.0000 is now 0.0000 - 0.0000 * -1.0000 = 0.0000
 Previous element (3,3) = -0.2000 is now -0.2000 - 0.0000 * -1.6000 = -0.2000

 Current Matrix:
    -1.0000     1.0000     1.0000  Inactive
     1.2000    -1.0000    -1.6000  Inactive
     0.4000     0.0000    -0.2000  Inactive


Inversion Complete
```

4. Microsoft® Visual Basic® Routine for In-Place Matrix Inversion

```
'(A) Sample calling routine for In-Place Subroutine

'Legend:
'iNmat      = Matrix size
'dblMatrix  = Matrix array

 Dim iNmat As Integer

'Sample matrix size = 3x3
 iNmat = 3

'Sample matrix elements
 ReDim dblMatrix(iNmat, iNmat) As Double

 dblMatrix(1, 1) = -1: dblMatrix(1, 2) = -1: dblMatrix(1, 3) = 3
 dblMatrix(2, 1) = 2: dblMatrix(2, 2) = 1: dblMatrix(2, 3) = 2
 dblMatrix(3, 1) = -2: dblMatrix(3, 2) = -2: dblMatrix(3, 3) = 1

'In-Place Inversion
 InPlace iNmat, dblMatrix()

'The matrix dblMatrix will now contain the inverse for post-processing.


'(B) In-Place Matrix Inversion Subroutine

Public Sub InPlace(iNmat As Integer, arrMatrix() As Double)

'In-Place Matrix Inversion Subroutine for Modified Gauss-Jordan Algorithm
'Author:  D. DasGupta, P.E., M.Tech., M.ASCE, MCP

'Legend:
'iNmat     = Matrix size
'arrMatrix  = Matrix array

 Dim iRow As Integer, jCol As Integer, iCycle As Integer
 Dim dPivot As Double, dAbsDiag As Double, iBig As Integer
 ReDim blnRowProcessed (iNmat) As Boolean

'Preset entire Processing Index vector to False (Unprocessed)
 For iRow = 1 To iNmat: blnRowProcessed(iRow) = False: Next iRow

'Loop over diagonal elements
 For iCycle = 1 To iNmat

'  Preset Pivot to Zero
   dPivot = 0!

'  Find the row number and value of largest diagonal element of
'  unprocessed rows (blnRowProcessed = False)

'  Loop over matrix rows
   For iRow = 1 To iNmat

'    If processing index for this row is False
     If blnRowProcessed(iRow) = False Then

'      Store ABS value of its diagonal element
       dAbsDiag = Abs(arrMatrix(iRow, iRow))
```

```
'     If ABS(Diagonal) > Current Pivot then
      If dAbsDiag > dPivot Then
'       Store current row number and diagonal value as pivot
'       Note:  Since all diagonal elements must be positive,
'       this test must succeed at least once
        iBig = iRow: dPivot = dAbsDiag
      End If

    End If

  Next

'  The number of active row with highest diagonal has been determined
'  and the value of its diagonal element has been stored as Pivot

'  Reset Index of the row with highest diagonal to Processed and its
'  diagonal element to Unity to reduce truncation error.

  blnRowProcessed(iBig) = True
  dPivot = arrMatrix(iBig, iBig)
  arrMatrix(iBig, iBig) = 1!

'  Divide the entire Pivotal row with Pivot
  For jCol = 1 To iNmat
    arrMatrix(iBig, jCol) = arrMatrix(iBig, jCol) / dPivot
  Next

'  This will make the Pivotal Row diagonal = Unity

'  Loop over all rows
  For iRow = 1 To iNmat

'    Skip the Pivotal row which has already been processed
    If iRow <> iBig Then

'     Retrieve the Pivotal column element of the current row and
'     reset it to zero

      dPivot = arrMatrix(iRow, iBig)
      arrMatrix(iRow, iBig) = 0!

'     Subtract from each element of the current row the
'       Pivotal column element times the Pivotal column element in
'       the row equal to current column
      For jCol = 1 To iNmat
        arrMatrix(iRow, jCol) = arrMatrix(iRow, jCol) - _
          dPivot * arrMatrix(iBig, jCol)
      Next

    End If

  Next

 Next

End Sub
```

5. Conclusion

During the early years of his career the author had developed a number of Structural Engineering softwares in India using his In-Place Inversion method as their analysis engine in FORTRAN language for mainframe applications. He later developed Microsoft™ QuickBasic, Compiled Basic, Visual Basic 6 (VB6) [4] [5] and Visual Basic for Applications (VBA) [6] versions of the inversion routine in the USA for PC-based engineering applications [7] [8] and spreadsheets, the last one of which has been presented here. The purpose of this article is to provide its readers with a ready-to-incorporate subroutine for VB6 softwares and MS-Excel™ spreadsheets and even if Visual Basic is phased out in future [9], it may still remain useful as a template for scientific and engineering software developers to create equivalent routines in other languages.

Acknowledgements

The author gratefully acknowledges his introduction to Matrix Algebra by late Prof. B. R. Seth, former Head of the Department of Mathematics, Indian Institute of Technology, Kharagpur, India; the guidance and encouragement from his ex-supervisor and mentor late K. Madhavan, former Deputy Director, Central Water and Power Commission, New Delhi, India and the valuable contributions and suggestions of his ex-colleague late M. R. Rao, former Mathematician/Programmer, Computer Center, Planning Commission, New Delhi, India.

About the Author

D. DasGupta is a retired Structural Engineer whose past positions included V.P., Development, LEAP Software, Inc., Tampa, FL; Principal Consultant, McDonnell Douglas Automation Co., St. Louis, MO; Senior Engineer, Parsons Brinckerhoff/Tudor, Inc., Atlanta, GA; Structural Engineer, LEAP Associates, Inc., Lakeland, FL and Assistant Director, Central Water and Power Commission, New Delhi, India. He received his M.Tech. degree from Indian Institute of Technology, Kharagpur, India and D.I.C. from Imperial College of Science and Technology, London, England. He is a Life Member of the American Society of Civil Engineers and former Professional Engineer registered in Florida, Pennsylvania and Texas. He is certified in AutomatH (FORTRAN) programming by Honeywell, Inc., a Microsoft-Certified Professional and MS-Office™ User Specialist/MS-Access™ Expert. During his career he has worked as a practicing engineer as well as the author of several computer software libraries for Hydraulics, Soil Mechanics, Structural Analysis and Engineering Design in India and the USA.

References

[1] Smith, W.A. (1986) Elementary Numerical Analysis. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 51-52.

[2] DasGupta, D. (2013) In-Place Matrix Inversion by Modified Gauss-Jordan Algorithm. *Applied Mathematics*, **4**, 1392-1396. https://doi.org/10.4236/am.2013.410188

[3] McFarland, T. (2007) The Inverse of an n × n Matrix. University of Wisconsin-Whitewater. http://math.uww.edu/~mcfarlat/inverse.htm

[4] Microsoft Press (1998) Microsoft Visual Basic 6.0 Programming Guide©.

[5] Holzner, S. (1998) Visual Basic 6 Black Book©. Coriolis Technology Press, Scottsdale, AZ, USA.

[6] Microsoft™ (2017) Language Reference VBA|MSDN. https://msdn.microsoft.com/en-us/vba/vba-language-reference

[7] Staff Reporter (1987) Roads, Bridges and Computers. *Roads & Bridges Magazine*, 48.

[8] Corporate Technology Information Services, Inc. (1990) Corporate Technology Directory. U.S. Edition, Corporate Technology Information Services, Inc., Woburn, MA, USA, 3-873.

[9] Microsoft™ (2017) Support Statement for Visual Basic 6.0 on Windows. https://docs.microsoft.com/en-us/visualstudio/vb6/vb6-support