EasyChair Preprint

№ 15878

Computing the Solution to N-Variables Simultaneous Equations

Ishraga Mustafa Awad Allam

March 17, 2025

COMPUTING THE SOLUTION TO N-VARIABLES SIMULTANEOUS EQUATIONS

Ishraga Mustafa Awad Allam[1]

[1] Information Technology & Network Administration, University of Khartoum, Khartoum City, Khartoum, Sudan
ishragaallam@Gmail.com OR imallam@uofk.edu

ABSTRACT

Mathematics is in everything we find in life. Mathematics is on our mental thinking. From our daily digit world of our lives nowadays: like cameras, televisions, computers and telephones to computing mathematics can solve a lot of things without common mistakes. Matrices with open sizes can resolve a large number of things; i.e., resolutions of games 3D, graphics, science, animations, economics, encryption, geography, physics, constructions, signal processing and pattern recognition. In statistics, matrices solving can calculate probabilities matrices like that of predictions, especially with large size one. Matrix or Matrices are used in optic science to account for refraction and reflection. The results obtained are satisfactory and reliable. They can be improved to solve simultaneous equations with n variables, to solve many applications. Research is needed to get all right results.

KEYWORDS

Mathematics, Applied Statistics, Matrices, Matrices' Inverses, Algorithms, Matrices' Determinants, Solution to n-variables Simultaneous Equations & Linear Algebra.

1. INTRODUCTION

Matrices have the following uses in our day-to-day life. Some of the uses of matrices in daily life are mentioned below:

1. *Encryption* – A very common use of matrix in daily life is encryption. We use it to scramble data for security purposes, and to encode and decode this data, we require matrices. There is a key that helps encode and decode data which is generated by matrices.
2. *Games especially 3D* – One application of matrices is in games. We use it to alter the object, in 3d space. They use the 3d matrix to 2d matrix to convert it into the different objects as per requirement.
3. *Economics and Business* -To study the trends of a business, shares, etc. and to create business models etc.
4. *Construction* – Another common application of matrices in real life is the construction sector. Have you seen some buildings that are straight but sometimes architects try to change the outer structure of the building? This can be done with matrices. A matrix is made of rows and columns you can change the number of rows and columns within a matrix. Matrices can help support various historical structures.
5. *Dance – contra dance* – It is used to organize complicated group dances.
6. *Animation* – It can help make animations more precise and accurate.
7. *Physics* – Matrices are applied in the study of quantum mechanics, electrical circuits, and optics. It helps in the calculation of battery power outputs, resistor conversion of electrical energy into another useful energy. Therefore, matrices play a major role in calculations. Especially in solving the problems using Kirchoff's laws of voltage and current. Matrices are also useful in electrical circuits and quantum physics. Moreover, matrices are used to solve AC network equations in electrical circuits.

8. *Geology* – Matrices are used for taking seismic surveys. [*1*]

2. Matrices

A matrix is a list of numbers, letters or symbols, called elements, written as a rectangular array with each element occupying a definite position, with the whole array enclosed by a bracket.

In many practical situations matrices provide a neat and concise way of storing information. What is more, if the information in one matrix is related to that in another, it is often possible to process the data and extract further information by the application of a few simple operations. [2]

Matrices' Algebra

```
#By: Ishraga Mustafa Awad Allam. On: 28 - 5 - 2022.
# Solution to n-variables Simultaneous Equations.
import math
n = int(input('Enter your number of variables for your equations : '))
M = [range(n), range(n)]
v = e = [range(n)]
S = [range(n), range(n)]
Fr = open('Solution.txt', 'w')
print("\n\r")

def SHOW(S):
        print("\n")
        Fr.write("\n")
        for x in range(0, n):
                for y in range(0, n):
                        print("   %16.3f"%S[x][y], end=' ')
                        Fr.write(str("   %16.3f"%S[x][y]))
                print("\n")
                Fr.write("\n")

A = [range(n), range(n)]
B = [range(n)]
prod = [range(n), range(n)]
```

2.1. Multiplication of Two Matrices

Summing each coefficient of each row of the matrix A, is multiplied to the corresponding coefficient of column one of matrix B. The resultant matrix is prod which is a global matrix.

```
def mult(A, B):
        print("\n\r Multiplying the TWO Matrices \n\r")
        Fr.write("\n\r Multiplying the TWO Matrices \n\r")
        for i in range(0, n):
                res = 0
                for j in range(0, n):
                        res = res + (A[i][j] * B[j])
                v.insert(i, res)
```

2.2. Determinant of the Matrix

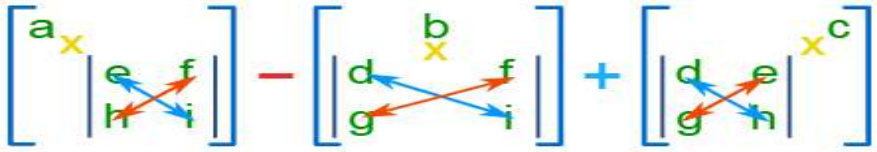
The function takes two variables, number of size of the matrix, and the matrix A. A Test is done on the size of the matrix as if it is a 2 x 2 one or 3 x 3 or more than 3 x 3. If it is a more than 3 x 3 matrix, the cofactors are inserted into a local 4-dimenisional variable, giving each coefficient of the matrix A, which is a local one, its corresponding cofactor matrix, by eliminating those along row and column of the coefficient. After that the function calls itself with the cofactors of the first row of the cofactor matrix but with a size less that that of the mother's one by one. It keeps calling [5] until the size is 3. The return resultant of the function is a local variable which with each call this value is pushed into a stack, It keeps calling until the size is 3, calculate it [6] and starts popping back from the stack. Also the value gets negative if the number of cofactor is dividable by 2. A python program is also available on [3] with the same algorithm. It gives the determinants with very large sizes.

(i). n = 2

[8], [9] and [7].

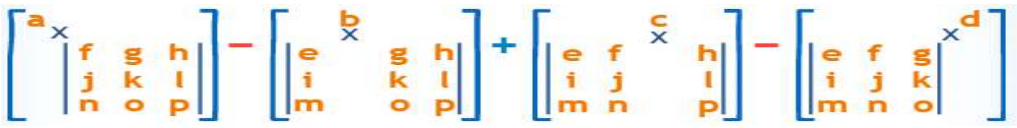
(ii)/ n = 3

$$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$



[7]

(iii) n > 3



$$|A| = a \cdot \begin{vmatrix} f & g & h \\ j & k & l \\ n & o & p \end{vmatrix} - b \cdot \begin{vmatrix} e & g & h \\ i & k & l \\ m & o & p \end{vmatrix} + c \cdot \begin{vmatrix} e & f & h \\ i & j & l \\ m & n & p \end{vmatrix} - d \cdot \begin{vmatrix} e & f & g \\ i & j & k \\ m & n & o \end{vmatrix}$$

[8]. [9] and [7]

```
def det(b, M):
        c = 0
        if(b == 2):
                determinant = (M[0][0] * M[1][1]) - (M[0][1] * M[1][0])
```

```
        if(b == 3):
                M1 = (M[0][0] * (M[1][1] * M[2][2] - M[2][1] * M[1][2]))
                M2 = (M[1][0] * (M[0][1] * M[2][2] - M[2][1] * M[0][2]))
                M3 = (M[2][0] * (M[0][1] * M[1][2] - M[1][1] * M[0][2]))
                determinant = M1 - M2 + M3
        if(b >= 4):
                C = [range(b), range(b), range(b), range(b)]
                for x in range(0, b):
                        C3= [range(b), range(b), range(b)]
                        for y in range(0, b):
                                p = 0
                                C2 = [range(b), range(b)]
                                for i in range(0, b):
                                        if(i != x):
                                                q = 0
                                                C1 = [range(b)]
                                                for j in range(0, b):
                                                        if(j != y):
                                                                C1.insert(q, M[i][j])
                                                                print("\n\r COEFFICIENTS of
cofactoring : [%d] [%d] [%d] [%d] : %8.3f"%(x, y, p, q, M[i][j]))
                                                                Fr.write(str("\n\r
COEFFICIENTS of cofactoring : [%d] [%d] [%d] [%d] : %8.3f"%(x, y, p, q, M[i][j])))
                                                                q = q + 1
                                                C2.insert(p, C1)
                                                p = p + 1
                                        C3.insert(y, C2)
                        C.insert(x, C3)

                determinant = 0
                for i in range(0, b):
                        c = c + 1
                        d = det(b - 1, C[i][0])
                        if(math.fmod(c, 2) == 0):
                                d = -d
                        determinant = determinant + (M[i][0] * d)
        return determinant

cofact = [range(n), range(n)]
```

2.3. Cofactors of the Matrix

A Cofactor, in mathematics, is used to find the inverse of the matrix, adjoined. The Cofactor is the number you get when you remove the column and row of a designated element in a matrix, which is just a numerical grid in the form of rectangle or a square.
The cofactor is always preceded by a positive (+) or negative (-) sign.[13]

Again like that of the determinant, but with the Test of the size of the matrix, this can be 2.

```
def cof(A):
        if(n == 2):
```

```
                cofact[0][0] = A[1][1]
                cofact[0][1] = -A[0][1]
                cofact[1][0] = -A[1][0]
                cofact[1][1] = A[0][0]
        else:
                C = [range(n), range(n), range(n), range(n)]
                for x in range(0, n):
                        C3= [range(n), range(n), range(n)]
                        for y in range(0, n):
                                p = 0
                                C2 = [range(n), range(n)]
                                for i in range(0, n):
                                        if(i != x):
                                                q = 0
                                                C1 = [range(n)]
                                                for j in range(0, n):
                                                        if(j != y):
                                                                C1.insert(q, A[i][j])
                                                                print("\n\r COEFFICIENTS of
cofactoring : [%d] [%d] [%d] [%d] : %8.3f"%(x, y, p, q, A[i][j]))
                                                                Fr.write(str("\n\r
COEFFICIENTS of cofactoring : [%d] [%d] [%d] [%d] : %8.3f"%(x, y, p, q, A[i][j])))
                                                                q = q + 1
                                                C2.insert(p, C1)
                                                p = p + 1
                                        C3.insert(y, C2)
                                C.insert(x, C3)
                        for x in range(0, n):
                                v2 = [range(n)]
                                for y in range(0, n):
                                        d = pow(-1, (x + y)) * det(n - 1, C[x][y])
                                        v2.insert(y, d)
                                cofact.insert(x, v2)

inv = [range(n), range(n)]
```

2.4. Inverse of the Matrix

As the determinant is calculated and the cofactors, it had became easier to calculate the inverse of the matrix A. The cofactors are given by calling the previous function cof(A).

```
def inverse(A):
        #cof(A)
        print("\n The Inverse of the Matrix \n")
        Fr.write("\n The Inverse of  the Matrix \n")
        for i in range(0, n):
                v = [range(n)]
                for j in range(0, n):
                        if(det(n, A) != 0):
                                v.insert(j, 1 / det(n, A) *  prod[i][j])
```

```
            inv.insert(i, v)
            prod.insert(i, v)
        SHOW(inv)
```

2.5. Solution to n-variables Simultaneous Equations

First the Matrix M is filled with coefficients and the Vector e is also is filled with components. The CoFactors of M are calculated. Then the Adjacent of M is calculated. Then from all this, the Inverse of M is calculated and is multiplied with the Vector e.

```
for i in range(0, n):
        l = [range(n)]
        for j in range(0, n):
            val1 = float(input(" COEFFICIENTS : %d by %d : "%(i + 1, j + 1)))
            l.insert(j, val1)
        M.insert(i, l)
print("\n")
Fr.write("\n")
SHOW(M)

for i in range(0, n):
        val2 = float(input(" Equating Value : %d : "%(i + 1)))
        e.insert(i, val2)
        Fr.write("%16.12f\n\r"%e[i])

 cof(M)
SHOW(cofact)
```

2.6. Transpose and Adjacent of the Matrix

Each coefficient of matrix A is replaced with the corresponding coefficient of the interchange of the number of column and row; i.e., the row becomes the column and vice versa. The resultant matrix is tran.

```
print("\n\r The Adjacent of the Matrix \n\r")
Fr.write("\n\r The Adjacent of the Matrix \n\r")
for i in range(0, n):
        v1 = [range(0, n)]
        v2 = [range(0, n)]
        for j in range(0, n):
            v1.insert(i, cofact[i][j])
            v2.insert(j, cofact[j][i])
        prod.insert(j, v1)
        prod.insert(i, v2)
SHOW(prod)
inverse(M)
mult(inv, e)

for i in range(0, n):
        print("%16.12f\n\r"%v[i])
        Fr.write("%16.12f\n\r"%v[i])
```

```
Fr.close()
```

3. Results

The logical of the procedures is correct, but the C program does not give accurate results. If a python program with the same logic, gives right results. [3] and [13]. Even with large n; i.e., the determinants for a matrix with n = 12, the identity one, gives the result of one. I did try the inverse of the matrix, to solve n variables simultaneous equations but did not work for me [15]. I was using Anaconda python version 3.7. More research need to be done.

4. Conclusion

Matrices, squared ones (n x n) can be computed by this study and may we obtain a finer results. I do hope this can make life, better. Easy procedures can be seen to resolve, big finer results. Determinants can be used to solve these equations (Cramer's rule), although other methods of solution are computationally much more efficient. Determinants are used for defining the characteristic polynomial of a matrix, whose roots are the eigenvalues. In geometry, the signed *n*-dimensional volume of a *n*-dimensional parallelepiped is expressed by a determinant. This is used in calculus with exterior differential forms and the Jacobian determinant, in particular for changes of variables in multiple integrals.

Acknowledgements

For Dr. Mohamed ElAmin ElTom; with sincere gratitude and appreciation:

I am so incredibly fortunate to have the benefit from his invaluable guidance, support and continual advice, without which this work would not have seen the light.

I am greatly indebted to Prof. Sharma from India, and Prof. Joseph A. Jervase, from South Sudan and who works at Oman.

Also my gratitude is extended to my family with all my heart and love, especially my late father.

References

[1] https://www.embibe.com/exams/where-are-matrices-used-in-daily-life/

[2] H. M. Kenwood & G. M. Staley (1984), Mathematics An Integrated Approach, Macmillan Education Limited, ISBN: 0-333-24581-4.

[3] Ishraga Allam (2022), n x n matrices' determinant, A Python3 program to compute Matrices (n x n) determinants. DOI: 10.13140/RG.2.2.12920.52483, https://www.researchgate.net/publication/357512465_n_x_n_Matrices'_determinant

[4] Ishraga Allam (2022), Solution to n variables Simultaneous Equations, DOI: 10.13140/RG.2.2.26358.27201, https://www.researchgate.net/publication/360912075_Solution_to_n_variables_Simultaneous_Equations

[5] Matrix(mathematics) https://en.wikipedia.org/wiki/Matrix_(mathematics)

[6] Determinant of A Matrix, https://www.mathsisfun.com/algebra/matrix-determinant.html#:~:text=The%20determinant%20helps%20us%20find,linear%20equations%2C%20calculus%20and%20more.Leendert Ammeraal (1986), C for Programmers, John Wiley & Sons Ltd., ISBN: 0 471 91128 3.

[7] L. Bostock, S. Chandler & C. Rourke, Further Pure Mathematics, Stanley Thomes (Publishers) 1984. ISBN: 0-85950-103-5.

[8] G. Hailey, Linear Algebra, Addison Wesley, 1961. Chapters: 1, 4 and 5.

[9] B. Nobbler and I. Daniel, Applied Linear Algebra, Chapters: 2 and 3.

[10] H. Schneider & G. P. Baker, Matrices and Linear Algebra, Holy Rinehart, 1968. Chapters: 1 and 2.

[11] Bjarne Stroustrup, AT&T, The C++ Programming Language, Addison-Wesley Longman Inc. (1997), Third Edition. ISBN: 0-201-88954-4.

[12] Michael A. Smith, Object-Oriented Software in C++, Chapman & Hall (1993), First Edition, ISBN: 0 412 55380 5.

[13] https://byjus.com/cofactor-formula/#:~:text=A%20Cofactor%2C%20in%20mathematics%2C%20is,of%20rectangle%20or%20a%20square.

[14] Ishraga Allam (2022), Factorizing an n x n matrix, My python3 program to compute Matrices (n x n) factor vector and the resultant matrix. DOI: 10.13140/RG.2.2.19631.41128 https://www.researchgate.net/publication/357517251_Factorizing_an_n_xn_matrix

[15] Determinant of A Matrix, https://www.mathsisfun.com/algebra/matrix-determinant.html#:~:text=The%20determinant%20helps%20us%20find,linear%20equations%2C%20calculus%20and%20more.Leendert Ammeraal (1986), C for Programmers, John Wiley & Sons Ltd., ISBN: 0 471 91128 3.

[16] Ishraga Allam (2019): **Computing (n x n) Matrices Algebra.**
DOI: 10.13140/RG.2.2.23727.46249
https://www.researchgate.net/publication/350194651_Computing_n_x_nMatrices'_Algebra

[17] Ishraga Allam (2021): **Computing n x n Matrices' determinant**
DOI: 10.13140/RG.2.2.12920.52483
https://www.researchgate.net/publication/357512465_n_x_n_Matrices'_determinant

[18] Ishraga Allam (2022): **Computing the Solution to n variables Simultaneous Equations.**
DOI: 10.13140/RG.2.2.26358.27201
https://www.researchgate.net/publication/360912075_Solution_to_n_variables_Simultaneous_Equations

Authors

(2010) Master in Computer and Mathematics Sciences:
(1993) Joint Honour on Computer and Statistics Sciences.