Geometry Algebra and Gauss Elimination method for solving a linear system of equations without division

Martin Cervenka
Faculty of Applied Sciences
University of West Bohemia
Pilsen, Czech Republic
cervemar@kiv.zcu.cz
0000-0001-9625-1872

***Abstract*—This paper aims to calculate the Gaussian elimination method without division operation, which is useful for cases where the division operation is considerably expensive, not optimised or inconvenient. To substitute the division, more multiplication steps are performed. The division is completely avoided, reaching only 7% longer execution time on a modern computer. Memory savings and also less multiplication has been reached in comparison to the state-of-the-art approach.**

***Index Terms*—Gaussian Elimination Division-free Linear equation system Geometry algebra**

I. Introduction

The Linear system of equations $\mathbf{Ax} = \mathbf{b}$ is defined by matrix $\mathbf{A}$ of size $N \times N$, where A_{ij} is a coefficient of i^{th} equation and j^{th} independent variable. The b_i is the i^{th} equation constant coefficient (also called right-hand side of the equation) and x_j are the dependent variable to solve. If the matrix is rectangular, the system is over-determined/under-determined. The over-determined system has more rows/equations than columns/independent variables, and the under-determined is the opposite. For the sake of simplicity, only square and regular (with matrix $\mathbf{A}$ of size $N \times N$ and order N) equation systems will be considered.

II. Gauss elimination method

Although the Gauss elimination method (GEM) with the complexity of $O\left(N^3\right)$ [1] is not optimal in theory [2], it is commonly used to solve reasonably small matrices, where its higher asymptotic complexity is suppressed. The basic idea behind the Gaussian elimination method is the matrix conversion from its initial form to the identity matrix form, using only the following operations:

- Multiplying a matrix row with a nonzero scalar value
- Adding a row to another row (or its arbitrary nonzero multiple)

To solve for vector $\mathbf{x}$, the $\mathbf{b}$ will be transformed in the same fashion as $\mathbf{A}$ (often, $\mathbf{b}$ is written inside $\mathbf{A}$, separated by a vertical line). Finding the inversion of the $\mathbf{A}$ matrix is the same problem as the linear equation solution using orthogonal unit vectors $\mathbf{b}$. All of them will form the identity matrix. Another way to describe the problem is to apply the same operations on the identity matrix, as there were applied on the original matrix $\mathbf{A}$.

The procedure is such that a combination of allowed operations zeroes the first column below the diagonal, then the second column, the third and so on. In the next step (backward cycle), the part above the diagonal in the last column is zeroed, then the second to last till the second column. The final step is multiplying all rows to obtain the ones on the diagonal.

A "shortcut" also solves the equation system but does not produce an inverse matrix. The shortcut lies in the backward cycle that can be solved directly from the last variable to the first. Further in the text, this shortcut will be called the "half-way" Gaussian elimination method, which will be stated explicitly. Otherwise, the complete Gaussian elimination method is meant.

In the case of the rectangular matrix, the approach would be the same as described here for the square matrix. However, the same restrictions apply to this approach as to the original GEM method (mainly the rank of the matrix should equal the number of variables in obtaining a single solution).

A. Partial pivoting

Partial pivoting is the computational step performed in the forward cycle to improve computational accuracy. When a new column is zeroed below the diagonal, the remaining rows (with higher indices than the column index) are swapped so that the pivot will have interesting properties. It is appropriate to select the pivot with the highest absolute value so that the roundoff errors will be less significant [3].

B. Algorithm

The Gaussian elimination algorithm is described in Listing 1. The `a` denotes the one-based indexed matrix $\mathbf{A}$, with the right-hand side column vector appended to the matrix (so its final dimension is $N \times (N + 1)$).

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic, project SGS-2022-015

979-8-3503-1034-4/22/$31.00 ©2022 IEEE

```
for k := 1 to n-1 do
{
    # Finds the maximum pivot, #
    # partial pivoting #
    ii := max_arg(abs(a[i, k]),i=k...n);
    if abs(a[ii, k]) <= eps
        ERROR ("Matrix_is_singular!");
        exit;
    swap_rows (k, ii); # swaps k, ii #
    # for all rows below pivot #
    for i := k + 1 to n do
    { # for all remaining elements #
      # in the current row #
        for j := k + 1 to n+1 do
        a[i, j] := a[i, j] - a[k, j]
            * (a[i, k] / a[k, k]);
        # fill lower triangular matrix #
        # with zeros if needed #
        a[i, k] := 0
    }
}
for i := n downto 1 do # backward loop #
{
    s:=0;
    for j := i+1 to k
        s := s + a[i,j] * x[j];
    x[i] := (a[i,n+1) - s) / a[i,i];
}
```

Listing 1. Gaussian elimination algorithm [4] (modified). The backward cycle does not edit the matrix but effectively calculates the result ("half-way" Gaussian elimination method).

III. Proposed approach

The main problem in the Gaussian elimination method is that the "Gaussian elimination to solve a system of n equations for n unknowns requires $\frac{n(n+1)}{2}$ divisions, $\frac{2n^3+3n^2-5n}{6}$ multiplications, and $\frac{2n^3+3n^2-5n}{6}$ subtractions" [5]. Be aware that the mentioned approach is just a "half-way" Gaussian elimination because it will create just the upper triangular matrix and solve it. In the case of the complete Gaussian elimination (and able to obtain matrix inverse), it will be even more costly than that. Luckily, the division count can be reduced or, even better, unnecessary.

Skala's article [6] proposed a projective extension of the Euclidean space to reduce the number of division operations; however, the right-hand side vector can be used instead of the homogenous coordinate for this purpose, discarding the homogenous coordinate multiplication step and saving some memory (not significant in asymptotic case, though). The only advantage is that there are two variables instead of a single one, allowing the possibility for better numerical stability.

The idea is that in each step, every other row then a selected one b can be multiplied by the pivot value from the row p, using the factor a_{pp}. Then, the row multiple can be added more simply, because the factor will be a whole number instead of a real one. The idea will work for rational numbers and irrational ones, as the irrational number can be used as a factor for other rows.

Operation	Complexity
Gauss. elim.	
/ (Division)	$\frac{1}{2}(N^3 - N)$
* (Multiplication)	$\frac{1}{2}(N^3 - N)$
- (Subtraction)	$\frac{1}{2}(N^3 - N)$
$\wedge$ (Bitwise AND)	0
$\vee$ (Bitwise OR)	0
$\oplus$ (Bitwise XOR)	0
Memory	N^2
Skala's approach	
/ (Division)	0
* (Multiplication)	$N^3 + 2N^2 - 3N$
- (Subtraction)	$\frac{3}{2}(N^3 + 2N^2 + 3N)$
$\wedge$ (Bitwise AND)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
$\vee$ (Bitwise OR)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
$\oplus$ (Bitwise XOR)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
Memory	$N^2 + N$
Proposed approach	
/ (Division)	0
* (Multiplication)	$N^3 + N^2 - 2N$
- (Subtraction)	$\frac{3}{2}(N^3 + 2N^2 + 3N)$
$\wedge$ (Bitwise AND)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
$\vee$ (Bitwise OR)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
$\oplus$ (Bitwise XOR)	$\frac{1}{2}(N^3 + 2N^2 - 3N)$
Memory	N^2

TABLE I
Comparison of operation count for all of the complete Gaussian elimination methods. The proposed approach improves the memory size and number of multiplication operations over Skala's approach [6].

A. Example of direct GEM

Let us assume an example linear equation system:

$$\begin{aligned} 2x_1 - x_2 &= 5 \\ 3x_1 - 4x_2 &= 6 \end{aligned} \tag{1}$$

The classical Gauss Elimination method would look like the following:

$$\left[\begin{array}{cc|c} 2 & 1 & 5 \\ 3 & 4 & 6 \end{array}\right] -\tfrac{3}{2}\mathrm{I} \sim \left[\begin{array}{cc|c} 2 & 1 & 5 \\ 0 & \frac{5}{2} & -\frac{3}{2} \end{array}\right] -\tfrac{2}{5}\mathrm{II} \tag{2}$$

$$\sim \left[\begin{array}{cc|c} 2 & 0 & \frac{28}{5} \\ 0 & \frac{5}{2} & -\frac{3}{2} \end{array}\right] \times\tfrac{1}{2} \times\tfrac{2}{5} \sim \left[\begin{array}{cc|c} 1 & 0 & 2.8 \\ 0 & 1 & -0.6 \end{array}\right]$$

In the first step, the $\frac{3}{2}$ of the first row is subtracted from the second row to eliminate A_{21}. In the second step, $\frac{2}{5}$ of the second row is subtracted from the first row to eliminate A_{12}. Finally, both rows are multiplied to get the identity matrix on the left. The solution to the problem can be found on the right.

B. Example of our approach

Let us assume the same example as it was in the III-A section with linear equation system $2x_1 - x_2 = 5$ and $3x_1 - 4x_2 = 6$. The issue is the division has to be done in every computation step. To avoid that problem, a different method is proposed, where each row $i, i \neq p$ is multiplied by the pivot (excluding the row containing the pivot) from a chosen row p (element a_{pp}) as follows:

$$\left[\begin{array}{cc|c} 2 & 1 & 5 \\ 3 & 4 & 6 \end{array}\right] \times 2 \sim \left[\begin{array}{cc|c} 2 & 1 & 5 \\ 6 & 8 & 12 \end{array}\right] -3\mathrm{I} \sim \tag{3}$$

$$\sim \left[\begin{array}{cc|c} 2 & 1 & 5 \\ 0 & 5 & -3 \end{array}\right] \times 5 \sim \left[\begin{array}{cc|c} 10 & 5 & 25 \\ 0 & 5 & -3 \end{array}\right] -\mathrm{II} \sim$$

$$\sim \left[\begin{array}{cc|c} 10 & 0 & 28 \\ 0 & 5 & -3 \end{array}\right] \times\tfrac{1}{10} \times\tfrac{1}{5} \sim \left[\begin{array}{cc|c} 1 & 0 & 2.8 \\ 0 & 1 & -0.6 \end{array}\right]$$

This method multiplies each row beforehand, so there is no need for division. The approach required only N divisions at the end to determine the result vector $(2.8, -0.6)$. The main issue here is that the matrix elements may grow/decay exponentially, making the approach useless for practical use on the computer. This phenomenon can be seen in Equ. (3), where diagonal elements grow, for bigger matrix will grow even more.

IV. Implementation

The proposal has been tested using the C++ programming language. Skala's row normalization [6] step has been adopted, which can also be done without division, mainly using bitwise operations.

```
#define FLT_MASK 0x800FFFFFFFFFFFFL
#define EXP_MIDDLE 0x3FF0000000000000L
// everything else than the exponent part
// of the "b" vector
unsigned right_data = a[i][N] & FLT_MASK;
//exponent part of the IEEE-754
// double precision variable
unsigned right_exp = (right_data ^
    a[i][N]) - EXP_MIDDLE;
for(int j=k;j<N; j++){
    //exponent part of each element
    // in the matrix row
    unsigned data = a[i][j] & FLT_MASK;
    //shift the exponent to the opposite
    //direction of the b vector exponent
    a[i][j] = data|
        ((data^a[i][j])-right_exp);
}
// make exponent of the
// "b" vector element "zero"
a[i][N] = right_data | EXP_MIDDLE;
```

Listing 2. Exponent normalization step, bitwise implementation, for every operation (OR, XOR, AND and subtraction), there is one of them performed inside the loop and one outside.

The `normalize` function deals with growing/decaying pivots in such a way that no division is needed. It is done by the exponent modification, using the same modification for each row element. As seen on Listing 2, the modification can be easily done on IEEE 754 floating point values, using just bitwise operations and a subtraction.

The `a` variable represents the matrix $\mathbf{A}$, last column of the `a` contains also right-hand side of the equation (column vector $\mathbf{b}$). `FLOAT_MASK` will mask bits belonging to the exponent part of the IEEE 754 floating point value, `EXPONENT_MIDDLE` is a value with zero exponent (exponent in IEEE 754 is shifted by 127).

V. Experimental results

Experiments were performed on the Hilbert matrix, where each element of the matrix is given by:

$$H_{ij} = \frac{1}{i + j - 1} \tag{4}$$

This particular matrix is well-known for its numerical instability during its inversion. The condition numbers of the matrix increase significantly with increasing N (see Tab. II). For the conditionality, 2-norm condition number of a matrix with respect to inversion was used ($||\mathbf{A}||_2 \cdot ||\mathbf{A^{-1}}||_2$).

Luckily, the Hilbert inversion matrix is well-known analytically [7] and can be evaluated by the Equ. (5), so the p-norm can be calculated more precisely using this inversion.

N	Conditionality	N	Conditionality
1	1.0000e+00	8	1.5257e+10
2	1.9281e+01	9	4.9315e+11
3	5.2406e+02	10	1.6024e+13
4	1.5513e+04	11	5.2227e+14
5	4.7660e+05	12	1.7515e+16
6	1.4951e+07	13	3.3441e+18
7	4.7536e+08	14	6.2008e+17

TABLE II
CONDITION NUMBER OF THE HILBERT MATRIX FOR GIVEN DIMENSION N.

The normalization step has been tested. It has been shown that the normalization step does not need to be performed in each computational step, but only if necessary (if the exponent is big/small enough to make it worthwhile). This approach is on average about 7% slower than the original, keeping all of the advantages (no division, no additive memory), see Fig. 1, Fig. 2 and Fig. 3.

$$H_{ij}^{-1} = (-1)^{i+j}(i+j-1) \binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2 \quad (5)$$

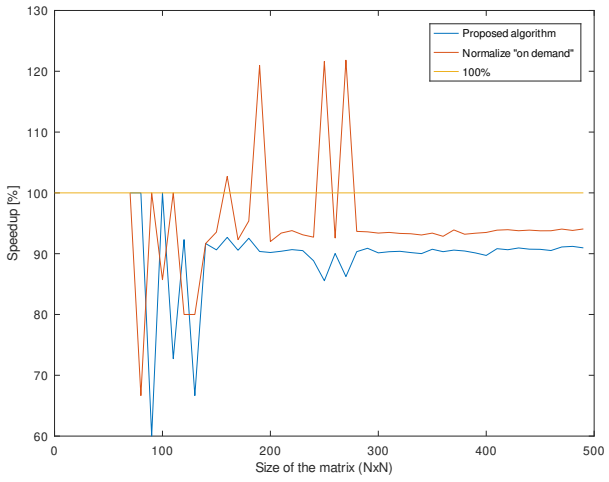

Fig. 1. Running speed of the proposed algorithm in proportion to original Gaussian elimination method. The Hilbert matrix inverse has been performed of the given size. The algorithm is slower than the original one, about 10%. It should be noted that the results may be inaccurate for higher N. The peaks are caused by the low computation time of all methods.

In Fig. 1 it can be seen that the proposed algorithm is about 10% slower than the original algorithm. The correctness of the execution can be seen in Fig. 2, where all of the algorithms provide results with (nearly) the same conditionality. Moreover, these results are the same as in Tab. II, caused by the fact that the conditionality of the matrix is equal to the conditionality of its inverse due to the commutativity property.

Forward cycle
Gaussian elimination

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] = a[i][j] - a[i][k]`
`  * a[k][j] / a[k][k]`

Skala's algorithm [6]

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] = a[i][j] * a[k][k] -`
`  a[i][k] * a[k][j];`
`a[i][HOMOG] = a[i][HOMOG]`
`  * a[k][k];`
`normalize(a, k+1, n)`

Proposed algorithm

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] := a[i][j] * a[k][k]`
`  - a[i][k] * a[k][j];`
`normalize(a, k+1, n)`

Backward cycle
Gaussian elimination

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] = a[i][j] - a[i][k]`
`  * a[k][j] / a[k][k]`

Skala's algorithm [6]

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] = a[i][j] * a[k][k]`
`  - a[i][k] * a[k][j];`
`a[i][i] := a[i][i] * a[k][k]`
`  - a[i][k] * a[k][i];`
`a[i][HOMOG] = a[i][HOMOG]`
`  * a[k][k];`
`normalize(a, k+1, n)`

Proposed algorithm

`for(int j=k+1; j<=n+1; j++)`
`  a[i][j] := a[i][j] * a[k][k]`
`  - a[i][k] * a[k][j];`
`a[i][i] := a[i][i] * a[k][k]`
`  - a[i][k] * a[k][i];`
`normalize(a, k+1, n)`

TABLE III
COMPARISON OF THE GAUSSIAN ELIMINATION ALGORITHM AND ITS MODIFICATIONS. IN THE BACKWARD CYCLE, THE DIVISION-FREE APPROACHES HAVE TO MODIFY THE DIAGONAL AS WELL.

The output error has been measured in Fig. 3. The normalized Frobenius norm of the difference of the matrix pair has been used:

$$L_2\left(\mathbf{X}\right) = \frac{\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(H_{ij}^{-1} - X_{ij}\right)^2}}{N^2} \tag{6}$$

The $\mathbf{H}^{-1}$ has been computed analytically; see Equ. (5).

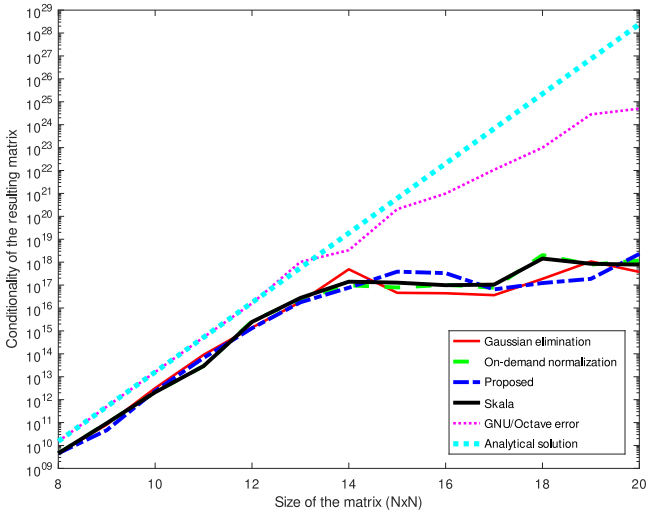

Fig. 2. Conditionality of the resulting inverse matrix. The results show that the conditionality is nearly the same for all algorithms. The differences between algorithms are pointless for $N > 12$ due to the high conditionality. The conditionality should be even higher, as the analytical solution shows. GNU/Octave inverse is approaching due to LU decomposition for matrix inversion.

The difference between the original Gaussian elimination method, Skala's modification [6], and the proposed one is shown on Tab. III.

VI. Conclusion & Future work

The Gaussian elimination method can be done without division and additive memory requirements, which is particularly useful in scenarios where the division operation is expensive. There will be no division if it is sufficient to obtain the result in projective space or N division in the Euclidean space. The improvement from Skala's publication [6] is that there is no need for storage for homogenous coordinates, saving memory for N variables and saving $N^2 - N$ multiplication operations. Despite these facts, the desired running time improvement has not been reached. Considering execution time, the division operation probably causes this to be no longer crucial.

Andrilli et al. state, "The partial pivoting technique is used to avoid roundoff errors that could be caused when dividing every entry of a row by a pivot value that is relatively small in comparison to its remaining row entries." [3]. Because division is unnecessary for this paper, the future task is to explore possibilities when partial pivoting will be avoided. It is also true that multiplication is still present. The true challenge will be to prevent the "nearly" zero factor, which may cause numerical instabilities.

This approach is particularly useful when the inversion is made not over a field of real numbers (matrix containing real numbers), but over a ring, where not all elements have multiplicative inverses, so dividing, in general, is impossible. However, the application of this approach is also future work to be done.

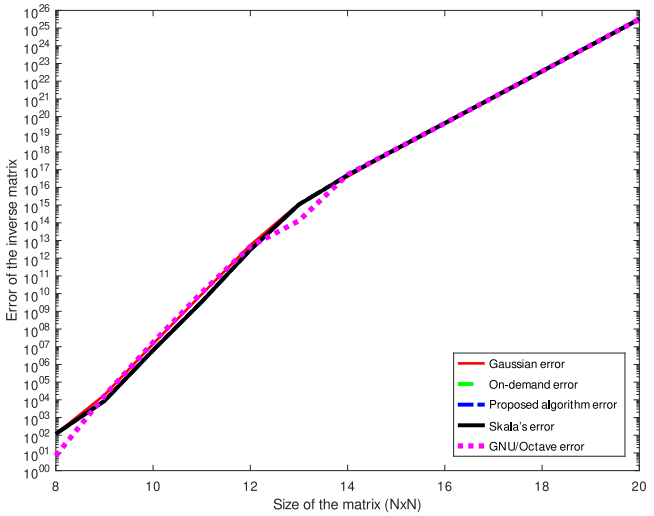

Fig. 3. The result difference between analytical inversion and a computed one. The differences between the algorithms are negligible.

Acknowledgments

The author thanks students and colleagues at the University of West Bohemia, Plzen. Special thanks belong to Vaclav Skala for his critical comments and discussion. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic, project SGS-2022-015.

References

[1] X. G. Fang and G. Havas, "On the worst-case complexity of integer gaussian elimination," in *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 28–31. [Online]. Available: https://doi.org/10.1145/258726.258740

[2] V. Strassen, "Gaussian Elimination is Not Optimal (1969)," in *Ideas That Created the Future: Classic Papers of Computer Science*. The MIT Press, 02 2021. [Online]. Available: https://doi.org/10.7551/mitpress/12274.003.0032

[3] S. Andrilli and D. Hecker, "Chapter 9 - numerical techniques," in *Elementary Linear Algebra (Fifth Edition)*, 5th ed., S. Andrilli and D. Hecker, Eds. Boston: Academic Press, 2016, pp. 607–666. [Online]. Available: www.sciencedirect.com/science/article/pii/B9780128008539000098

[4] R. Bronson and G. B. Costa, "Chapter 3 - the inverse," in *Matrix Methods (Fourth Edition)*, 4th ed., R. Bronson and G. B. Costa, Eds. Academic Press, 2021, pp. 93–129. [Online]. Available: www.sciencedirect.com/science/article/pii/B9780128184196000034

[5] N. S. Rani, "Nondeterministic procedure of solving simultaneous equations," pp. 49–54, 02 2013. [Online]. Available: www.researchinventy.com/papers/v2i3/G023049054.pdf

[6] V. Skala, "Modified gaussian elimination without division operations," *AIP Conference Proceedings*, vol. 1558, no. 1, pp. 1936–1939, 2013. [Online]. Available: aip.scitation.org/doi/abs/10.1063/1.4825912

[7] K. Neshat, "Proof that the hilbert matrix is invertible with integer entries," 05 2020. [Online]. Available: www.researchgate.net/publication/341215305_Proof_that_the_Hilbert_Matrix_is_Invertible_with_Integer_Entries