BAREKENG: Journal of Mathematics and Its Applications
March 2026 Volume 20 Issue 1 Page 0743–0754
P-ISSN: 1978-7227 E-ISSN: 2615-3017
https://doi.org/10.30598/barekengvol20no1pp0743-0754

SYMBOLIC COMPUTATION APPROACH TO REDUCE ROUNDING ERRORS IN NUMERICAL OPERATIONS USING RYACAS

I Gusti Agung Anom Yudistira[1*], **Kie Van Ivanky Saputra**[2]

[1]*Statistics Department, School of Computer Science, Universitas Bina Nusantara*
Jln. K H. Syahdan No. 9, Kecamatan Palmerah, Jakarta Barat 11480Jakarta, 11480, Indonesia

[2]*Department of Mathematics, Universitas Pelita Harapan*
Jln. M. H. Thamrin Boulevard 1100 Lippo Karawaci, Tangerang, 15811, Indonesia

*Corresponding author's e-mail: * **i.yudistira@binus.ac.id***

Article Info

Article History:

Received: 16th May 2025
Revised: 28th July 2025
Accepted: 30th July 2025
Available online: 24th November 2025

Keywords:

Precision errors;
R programming;
Ryacas;
Yacas;

ABSTRACT

Computational operations on computers produce only approximations due to the limitations of numerical representation, finite precision arithmetic, and hardware constraints. For simple calculations, these errors are usually negligible. However, in a sequence of numerical computations, they can propagate and accumulate, leading to significant inaccuracies and becoming a critical issue, where small errors can have substantial consequences. R, like other programming languages designed for numerical computations, is not immune to precision errors. One approach to this issue is to preserve exact values throughout calculations. In R, there are several packages, such as Ryacas and Ryacas0 enable symbolic computation, which allow true values to be maintained during operations. In this paper, we propose an application of computational techniques that effectively eliminates precision errors arising from numerical calculations. We developed a user-defined function for solving linear systems using Gauss-Jordan row elementary operations. We first developed a function to solve linear systems without using the Ryacas package, named OBE.R, and another function with the same purpose but now using Ryacas, named yac-OBE.R. These two functions are compared, and as expected, the latter eliminates numerical precision errors; hence, the accuracy is one hundred percent improved. Additionally, this study is limited to solving linear systems with a unique solution and does not discuss cases with multiple solutions or no solution. Also, symbolic computation as implemented via Ryacas typically requires more processing time.

How to cite this article:

I. G. A. A. Yudistira and K. V. I. Saputra, "SYMBOLIC COMPUTATION APPROACH TO REDUCE ROUNDING ERRORS IN NUMERICAL OPERATIONS USING RYACAS", *BAREKENG: J. Math. & App.*, vol. 20, iss. 1, pp. 0743-0754, Mar, 2026.

Journal homepage: *https://ojs3.unpatti.ac.id/index.php/barekeng/*
Journal e-mail: *barekeng.math@yahoo.com; barekeng.journal@mail.unpatti.ac.id*

Research Article · **Open Access**

1. INTRODUCTION

Real numbers are represented in a computer using floating-point numbers, which means they are stored in a finite format that approximates exact values using a fixed number of bits. Due to this finite precision, floating-point arithmetic is susceptible to rounding errors, truncation, and loss of significance, particularly in iterative or large-scale computations. Therefore, controlling and optimizing these errors in numerical computations is very important to get the desired precision [1]. R, as a programming language designed for numerical analysis, provides powerful tools for computation [2]. However, it is not immune to truncation errors, which can lead to significant inaccuracies. To address this, symbolic computation software such as Yacas (Yet Another Computer Algebra System) [3] and Wolfram Alpha [4][5] can be used. While Wolfram Alpha offers more advanced features, it comes at a cost, making it less accessible for R users, who require symbolic computation. The following simple example illustrates how numerical operation errors occur due to floating-point precision limitations in R [6][7].

A simple numeric example demonstrates that an arithmetic operation, which theoretically should yield zero, instead produces a very small nonzero value. This discrepancy arises from floating-point representation and rounding errors [8]. Although such errors may appear negligible in isolation, they can accumulate and significantly affect results in large-scale computations.

Another important issue occurs when direct equality comparisons are used. A condition that should logically evaluate as true may instead be evaluated as false due to tiny precision errors, leading to logical mistakes in programs. Similarly, R may fail to compute the exact inverse of certain matrices because fractional values such as 4/9 or 1/3 are automatically converted into decimal approximations. When the matrix is multiplied by its inverse, the result deviates slightly from the identity matrix, producing small numerical errors, such as -1.110×10^{-16} and 5.551×10^{-17}, instead of exact zeros [9]. While these errors are small, they can accumulate in iterative computations, as happened in the Gauss-Jordan row elimination [10]. For computations requiring high precision, these numerical errors can be problematic.

Errors due to numerical computation can be fatal, especially when the program involves conditional branching (if-else statement). A condition in the if-else statement that should be TRUE may instead be evaluated as FALSE due to numerical computation errors [6]. This occurs due to rounding errors in numerical computation. The output of a matrix operation can be incorrect due to these errors. Although the error may be small, if the computation involves multiplication with very large numbers, the error can propagate significantly. An example of this can be seen in Fig. 2. Therefore, it is crucial to develop techniques to mitigate such issues.

This paper focuses on mitigation techniques for floating-point errors that arise in R computations using the Ryacas package. The selected computational case is solving a system of linear equations. The solution to this system is not merely obtained using the built-in solve function in R, but through a step-by-step and interactive computation. Thus, the output of this research is a custom R function for performing interactive symbolic computations step by step using the Ryacas package. This R function is an improvement over conventional interactive numerical computation functions that are prone to floating-point errors [11].

In this research, we propose a solution to mitigate and even eliminate errors arising from numerical operations, particularly for R users. This paper aims to: 1) Develop a procedure in R that significantly reduces or eliminates numerical errors, particularly in row operations used to solve linear systems. 2) Explore the capabilities of the Ryacas and Ryacas0 packages in handling symbolic computations within R.

2. RESEARCH METHODS

This research uses literature and quantitative methods with descriptive and explanatory approaches. The following are the detailed steps of our research [12][13]:

1. Literature Review:
   a. Review up-to-date references about Yacas and Ryacas.
   b. Review up-to-date references about R programming features to get an interactive output.
2. System Design: Construct a pseudo-code describing the system we want to model.
3. Implementation: Build R scripts as simple as possible.

4. Analyze the outputs and validate.
5. Go back to step 3 if the output does not satisfy the requirement.

In the first two steps, we will thoroughly explore the Yacas and Ryacas libraries. We will apply these libraries to execute row elementary operations. Although R has a built-in function to solve linear systems, the primary goal of this research is not only to find the solution but to observe the step-by-step process of obtaining it. This allows for careful identification of errors that may arise at each stage. In the third step, we will construct a flowchart and write pseudocode to clearly explain our methodology. We will then develop the corresponding R scripts in Step 4. After the scripts are created, Step 5 will involve running the code, verifying its logical consistency, and analyzing the outputs. Any errors produced will be investigated and corrected as needed. Finally, we will compare our results with those obtained by other methods. We will revisit Step 3 if necessary.

This research limits its computation to only obtaining a numerical solution of a mathematical operation and obtaining an approximate value, which is closer to the exact value. We focus on developing algebraic computation methods, particularly for solving systems of linear equations. In this study, we consider only systems that have a unique solution, and ignore cases where the system has either infinitely many solutions or no solution. The R script is designed to facilitate interactive computation, allowing users to observe each step of the operation [14]. This approach can also be applied to observe the accumulation of numerical errors at each step. The outcome of this research is an R script, which enables anyone interested to explore, modify, and extend the method for further applications.

As previously mentioned, the algebraic operations discussed in this paper are those involved in obtaining solutions for linear systems. The computation does not discuss whether the system possesses a unique solution, multiple solutions, or no solution. Our script is designed to provide an interactive, step-by-step computation [12], with the primary goal of enabling users to identify errors that may arise at each stage of the calculation. We hope that the R script we have produced will be useful and can be further explored and improved as needed.

Additionally, we note that symbolic computation, as implemented via Ryacas, typically requires more processing time compared to standard numerical methods in R [15]. This is due to the overhead of manipulating algebraic expressions rather than raw numbers. However, the symbolic approach provides clearer insight into each computational step, which is valuable for identifying rounding errors and understanding the structure of the solution. Numerical methods, while faster and more scalable, may obscure intermediate steps and accumulate errors silently. Our approach balances these aspects by using symbolic computation selectively, where transparency and error tracking are most critical.

Symbolic computation is also particularly valuable in domains where accuracy and transparency of intermediate steps are critical. For instance, in cryptography, symbolic manipulation ensures precise algebraic transformations that underpin secure encryption algorithms [16]. In aerospace engineering, symbolic methods are used to derive exact control laws and verify system stability analytically [17]. Financial modeling also benefits from symbolic approaches, especially in sensitivity analysis and formula validation, where rounding errors can lead to significant misinterpretations [18]. These examples illustrate the broader relevance of our symbolic approach beyond numerical linear algebra.

While this study focuses on linear systems with unique solutions, the symbolic framework employed—particularly via the caracas package—can be extended to higher-dimensional systems and non-linear equations [19]. Symbolic matrix operations and equation manipulation are inherently scalable, and symbolic representations of non-linear models can support exact transformations and analytical insights. These extensions, while beyond the current scope, offer promising directions for future work.

3. RESULTS AND DISCUSSION

Suppose N is the true value of a certain quantity and n is its approximate value, then

$$N = n + e, \tag{1}$$

where e is the error term. Numerical computations using computers, especially those involving real numbers, will always produce approximations and therefore produce errors. One of the primary reasons for this is the floating-point representation of real numbers, which has a finite precision. In the following illustration, we

show how an error appears in a very simple computation in R. We use the default parameter options(digits = 7).

```
> x <- 1/3 + 1/2
> x
[1] 0.8333333
> x - 5/6
[1] -1.110223e-16
```

Figure 1. R Coding to Illustrate How Errors Arise in Very Simple Calculations in R

Based on Fig. 1, it can be seen that the result of summing two real numbers, 1/3 and 1/2, which yields 0.8333333—an approximation of 5/6. The error of this approximation is $-1.110223 \times 10^{-16}$. If we are not careful in applying arithmetic operations, the successive computations can cause the results to deviate significantly from the desired results. This happens because R, like most programming languages, relies on approximate numerical representations rather than exact values [20]. Suppose we continue the operation as follows:

```
> x <- (1/3 + 1/2 - 5/6)
> y <- x * 10^16
> y
[1] -1.110223
```

Figure 2. R Coding to Illustrate How Errors Can Become So Wildly Distorted

In Fig. 2, it is shown that the result of the computation is -1.110223, which deviates significantly from 0, the expected exact result. If the value of x from the previous computation can be preserved with greater precision, a more accurate outcome could be achieved.

A library that allows symbolic computation in R, called Ryacas, has been developed by Mikkel Andersen [21]. Ryacas serves as an interface between R and Computer Algebra software called Yacas (*Yet Another Computer Algebra System*), allowing users to perform algebraic manipulations and exact arithmetic within the R environment. With this library, R users can send unprocessed Yacas strings and other R objects to the Yacas process and receive exact results in return. Ryacas also supports high-precision arithmetic and symbolic computations within R. As a result, R users can preserve the exact value throughout their computations, minimizing numerical errors as long as arithmetic operations are performed.

We will revisit our previous example, which demonstrated numerical errors, and recompute it using the Ryacas and Ryacas0 libraries (an earlier version of Ryacas) as follows

```
> library(Ryacas)
> library(Ryacas0)
> rm(list = ls())
> "(1/3 + 1/2 - 5/6) * 10^16" %>%
+       yac_expr() %>% eval()
[1] 0
```

Figure 3. R Code to Illustrate the Use of the Ryacas Package to Correct Floating Point Errors

Based on Fig. 3, it is shown that the computation now appears correct, as the resulting error is zero. More details of Ryacas can be accessed at https://r-cas.github.io/ryacas/.

Solving a system of linear equations using Gaussian elimination involves a sequence of arithmetic operations, including row exchanges, row multiplications by a constant, and row additions with a multiple of another row [22]. These operations are performed consecutively, providing a clear illustration of how numerical errors can accumulate at each step [23]. The more steps required, the greater the accumulation of errors, particularly in large-dimensional systems [24].

In this paper, we are going to design an R script that solves systems of linear equations interactively and step by step. While R provides the built-in solve() function for this purpose, it does not allow users to

closely observe each arithmetic operation performed during the computation. Additionally, modifying or improving the built-in function to improve numerical accuracy (minimizing errors) is difficult.

The user-defined function called OBE() has been developed to solve systems of linear equations interactively and step by step. This function is based on the Gauss-Jordan elimination method, allowing R users to choose which row operation to perform at each step. The final output is a vector containing the numerical solution of the linear system [11]. The OBE() function will be used to solve systems of linear equations, and then we will evaluate the accumulated error resulting from consecutive arithmetic operations. The full script of OBE() is provided in Appendix 1 (OBE).R. Additionally, the corresponding pseudocode can be accessed (pseudo_code.pdf), which outlines the step-by-step logic of the computational process.

Suppose the system has dimension k, meaning the solution vector is also k-dimensional. Consequently, we will obtain k approximate solutions. In this research, we use systems with known exact solutions, allowing us to compute the numerical error. The error metric we adopt is the maximum relative error, defined as the largest relative error among all elements in the solution vector (e_{max}).

$$e_{max} = \max_i \left\{ \left| \frac{\left(x_{true,i} - x_{approx,i}\right)}{x_{true,i}} \right| \times 100\% \right\} \tag{2}$$

The above function will be used as long as the true solution is not zero. Theoretically, the maximum relative error (e_{max}) can be reduced if we preserve exact values at each step, eliminating truncation errors caused by floating-point representation.

The OBE function is designed with the following three core functions[25][26]:

1. swap() – Performs row exchange.
2. times() – Multiplies a row by a constant
3. multiple() – Adds a multiple of one row to another.

Each of these functions updates the global variable x, ensuring that every change in x is immediately reflected without the need to store intermediate results in a separate object or variable. This approach maintains continuity in computations.

As an illustration of the OBE() function, we consider the following system:

$$\begin{aligned} 600x + 800y &= 200 \\ 30.001x + 40.002y &= 10.001 \end{aligned} \tag{3}$$

Initially, we will define the numerical precision up to 15 significant digits, which is commonly used in floating-point arithmetic. The maximum precision allowed is 20 significant digits, options(digits = 15). We are then going to define an augmented matrix derived from the above linear system. We can define this matrix using R.

```
> Av <- matrix(c(600,    800,    200,
+                30001/1000, 40002/1000, 10001/1000),
+              nrow=2, byrow=TRUE)
> print(Av, digits = 15)
        [,1]    [,2]    [,3]
[1,] 600.000 800.000 200.000
[2,]  30.001  40.002  10.001
```

Figure 4. R Code to Create Av Augmented Matrix

The matrix **Av**, as illustrated in Fig. 4, is defined as the augmented matrix corresponding to the system in Eq. (3), where the right-hand side of Eq. (3) is represented by the third column of **Av** (**Av**[,3]). This matrix serves as the input for the OBE() function and is stored in the object OprRow, as defined below.

```
> OprRow <- OBE(x = Av, row = TRUE)
```

Figure 5. The Use of the OBE Function with Input is the Av Matrix and Stored in the OprRow Object

The use of the OBE function, illustrated in Fig. 5, involves a Boolean argument called row. When row is set to TRUE, the function performs row elimination. To obtain the solution of the system in Eq. (3), we

apply the three fundamental row elementary operations previously defined (swap(), times(), and multiple()). The R scripts implementing these operations can be found in Appendix 2.R.

At the final step, we obtain a solution as shown in Fig. 6 below.

```
> solution <- OprRow$output()$Equivalent.Matrix[,3]
> solution
[1] -1.00000000000710  1.00000000000533
```

Figure 6. The Result of the Last Step of Elementary Row Operations Using the OBE Function

The solution to the system of linear equations presented in Eq. (3) is shown in Fig. 6, yielding $x = -1.00000000000710$ and $y = 1.00000000000433$, whereas the exact values are $x = 1$ and $y = 1$. The true value of the solution is shown in Fig. 7 below.

```
> x <- -1; y <- 1
> true.val <- c(x, y) # [1] -1  1
> true.val
[1] -1  1
```

Figure 7. Defining True Values with R Code

The R code used to define the exact values of x and y, created as a vector and stored in an object named true.val, is shown in Fig. 7.

The absolute error is computed as follows (Fig. 8 below).

```
> Error <- abs(solution - true.val)
> Error
[1] 7.10187464392220e-12 5.32907051820075e-12
```

Figure 8. The Absolute Error

The R code used to compute the absolute error with the built-in abs function is presented in Fig. 8. The absolute error values for the solutions of x and y are $7.10187464392220 \times 10^{-12}$ and $5.32907051820075 \times 10^{-12}$, respectively. These values are stored in a numeric vector object named Error. The maximum error is shown in Fig. 9 below,

```
> max(Error)
[1] 7.1018746439222e-12
```

Figure 9. The Absolute Maximum Error

The R code used to obtain the maximum error by using the built-in max function is displayed in Fig. 9, yielding a value of $7.1018746439222 \times 10^{-12}$.

The calculation of the relative maximum true error in R is illustrated in Fig. 10 using the following code.

```
> # absolute relative maximum error
> rel <- abs((solution - true.val) / true.val) * 100
> paste0(max(rel), " %")
[1] "7.1018746439222e-10 %"
```

Figure 10. The Absolute Relative Maximum Error

The calculation of the relative maximum true error in R is shown in Fig. 10. This error is obtained by dividing the maximum true error by the true value and then multiplying the result by 100% [1]. The final result is stored in an object named rel, with a value of $7.1018746439222 \times 10^{-10}$%. The paste0 function is used to append the percentage symbol to the numeric value stored in rel, producing a readable percentage format.

Next, we will provide an alternative solution to the above step, which will preserve the exact value on each step of the row elementary operations. The function OBE() will be modified by applying methods provided by the packages Ryacas and Ryacas0. Objects such as vectors and matrices will be converted to "yac_symbol" objects using the ysym function provided by the Ryacas library. This conversion allows computations to be performed symbolically, eliminating truncation errors and ensuring that exact arithmetic is maintained throughout the row operations.

The modified OBE() function will be renamed yac_OBE(), which needs input matrices that have already been converted into "yac_symbol" objects. This ensures that all computations are performed symbolically, preserving exact values throughout the operations. The full R script for yac_OBE() can be found in Appendix 3 (yac_OBE).R

Initially, the matrix is constructed as shown in Fig. 11.

```
> library(Ryacas)
> AvCh <- matrix(c("600",    "800",    "200",
+                  "30001/1000", "40002/1000", "10001/1000"),
+                nrow=2, byrow=TRUE)
> yac.x <- AvCh %>% ysym()
> yac.x
{{       600,        800,        200},
 {30001/1000,  20001/500, 10001/1000}}
```

Figure 11. R Code to Create the Augmented Matrix AvCh and then Convert it into the yac_Symbol Objects.G

Similar to Fig. 4, Fig. 11 presents the R code used to define the augmented matrix derived from Eq. (3); however, in this case, the matrix elements are character-type and stored in an object named **AvCh**, which is then passed to the ysym() function from the Ryacas package. The ysym() function requires input elements to be of character type. The ysym() function available in the Ryacas package is used to convert the **AvCh** matrix into a yac_symbol object. The result of the conversion is stored in the yac_x object. Now the **yac_x** matrix is a matrix with exact elements (avoiding rounding). Matrix **yac.x** is the 2×3 augmented matrix and is the input of the function yac_OBE.

Here we are going to show the series of row operations. First, the yac_OBE function will be called and is stored in the OprRow object (Fig. 12).

```
> OprRow <- yac_OBE(x = yac.x, row=TRUE)
```

Figure 12. R Code to Call the yac_OBE Function and Store it in the OprRow Object

The use of the user-defined function yac_OBE is demonstrated in Fig. 12. The complete R code for this custom function is available via the link in Appendix 3 (yac_OBE).R. This function takes two inputs: x and row. The input x refers to the augmented matrix yac_x, and when the value of row is TRUE, the operation performed is row processing. The output of this function is stored in an object named OprRow, which contains three functions: swap(), times(), and multiple(). To invoke the multiple() function, for example, the command OprRow$multiple() is used. The output of this function can then be passed interactively to the same or other functions in succession, eventually producing the desired result in the form of an equivalent matrix. This programming approach facilitates step-by-step row operations until the final result is obtained.

Step 1. We are going to multiply the first row by 1/600 such that the element in the first row and in the first column is 1. Fig. 13 below presents the R code for the computation using the yac_OBE function.

```
> # Step 1: i = 1, and k = "1/600"
> i = 1; k = "1/600"
> OprRow$times(i, k)
{{         1,        4/3,        1/3},
 {30001/1000,  20001/500, 10001/1000}}
```

Figure 13. R Code to Multiply the First Row of the yac.x Matrix by k = 1/600, while Maintaining the Result in Symbolic Form (Exact Value)

The R code for step 1 is presented in Fig. 13, where the scalar multiplier $k = \text{“}1/600\text{”}$ is assigned a character type as input to preserve its symbolic form. Ryacas requires inputs to be of character type when performing matrix multiplication with a scalar. The resulting output remains symbolic (i.e., exact), which ensures accuracy and avoids errors due to rounding. The result is the following matrix,

$$\begin{pmatrix} 1 & 4/3 & 1/3 \\ 30001/1000 & 20001/500 & 10001/1000 \end{pmatrix}. \tag{4}$$

Step 2. We are going to add the multiple of row 1 (by $k = -30001/1000$) to row 2. As a result, the element in row 2 and column 1 will be 0. Fig. 14 below presents the R code for this operation.

```
> # Step 2: i = 1, j = 2, and k = "-30001/1000"
> i = 1; j = 2; k = "-30001/1000"
> OprRow$multiple(i, j, k)
{{     1,    4/3,    1/3},
 {     0, 1/1500, 1/1500}}
```

Figure 14. R Code, to Add the Multiple of Row 1 (by $k = -30001/1000$) to Row 2

The objective of the operation in the second step is to obtain a matrix equivalent to the one obtained in the previous step, where the element in row 2, column 1 is 0, while preserving the exact form of each element in the matrix.

The R code for step 2 is presented in Fig. 14, where the result from step 1 is continued by multiplying row 1 by $k = -30001/1000$ and adding it to row 2. The function used is OprRow$multiple with inputs $i = 1$, $j = 2$, and $k = \text{"}-30001/1000\text{"}$. The output of this function does not need to be stored in a specific R object, as it is automatically saved and can be used in the subsequent OprRow operations. The result of step 2 is the following matrix,

$$\begin{pmatrix} 1 & 4/3 & 1/3 \\ 0 & 1/1500 & 1/1500 \end{pmatrix}. \tag{5}$$

Step 3. We are going to multiple row 2 by k = 1500, and as a result, the element in row 2, column 2 will be 1. Fig. 15 below presents the R code for this operation. The objective of the operation in the third step is to obtain a matrix equivalent to the one obtained in the previous step, where the first nonzero element in row 2 is set to 1, while preserving the exact form of each element in the matrix.

```
> # Step 3: i = 2, and k = "1500"
> i = 2; k = "1500"
> OprRow$times(i, k)
{{  1, 4/3, 1/3},
 {  0,   1,   1}}
```

Figure 15. R Code, to Multiple Row 2 by k = 15002

The R code used to perform step 3 is displayed in Fig. 15. In this step, the matrix obtained from step 2 is processed using the function OprRow$times(), with inputs $i = 2$ and $k = 1500$. The resulting output is the following matrix,

$$\begin{pmatrix} 1 & 4/3 & 1/3 \\ 0 & 1 & 1 \end{pmatrix}. \tag{6}$$

Step 4. We are going to add the multiple of row 2 to row 1 (the multiplication factor is $k = -4/3$). Fig. 16 below presents the R code for this operation.

```
> # Step 4: i = 2, j = 3, and k = "-4/3
> i = 2; j = 1; k = "-4/3"
> OprRow$multiple(i, j, k)
{{ 1,  0, -1},
 { 0,  1,  1}}
```

Figure 16. R Code to Add the Multiple of Row 2 to Row 1 (Multiplication Factor is $k = -4/3$)

The R code for performing step 4 is presented in Fig. 16, where the interactive function OprRow$multiple is applied with inputs $i = 2, j = 1$, and $k =$ "4/3". The result is the following matrix.

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \end{pmatrix}. \tag{7}$$

The operation in step 4 is the final operation, in which we obtain an equivalent system of linear equations that is easy to solve. Based on the output matrix from the R code shown in Fig. 16 above, the equivalent system of linear equations is as follows:

$$\begin{aligned} x + 0y &= -1 \\ 0x + y &= 1 \end{aligned} \tag{8}$$

It is clear that we obtain the solution directly, namely $x = -1$ and $y = 1$. Step 5 below provides the R code to extract these solution values.

Step 5. In this final step, we have already obtained the solution of the linear system as shown in the last column of the matrix in the last step.

```
> # Step 5: The final result is the 3rd column of the Equivalent.Matrix
> yac_solution <- OprRow$output()$Equivalent.Matrix[,3] %>%
+   yac_expr() %>% eval()
> yac_solution
[1] -1  1
```

Figure 17. The R Code to Obtain the Final Solution of the Linear Equation System Expressed by Eq. (3)

The R code involving the function OprRow$output, used at the end of the operation, is presented in Fig. 17. This function is employed to generate the final output of the elementary row processing. Two matrices are produced: the original matrix (Original.Matrix), which refers to the initial augmented matrix used as the input (yac_x), and the equivalent matrix (Equivalent.Matrix), which is the result of the final row operations. The final solution of the linear system defined by Eq. (3) is provided by the last column—column 3—of the equivalent matrix. Thus, to obtain the solution, the command OprRow$output()$Equivalent.Matrix[,3] is used, followed by the functions yac_expr() and eval(). The result of this command is stored in an object named yac_solution. At the bottom of Fig. 17, the resulting numeric vector is displayed, consisting of the values −1 and 1, corresponding to $x = -1$ and $y = 1$. As one can see, the obtained solution is exactly the same as the true solution, which is $x = -1$ and $y = 1$. Therefore, the error is 0.

4. CONCLUSION

To achieve higher precision in approximation and minimize floating-point / rounding error, this study employs the Ryacas package to retain exact values in R computations. We compared the standard OBE() function with the modified yac_OBE(), which integrates symbolic processing via Ryacas. The yac_OBE() function consistently preserves exact values, unlike OBE(), which introduces a relative error of 7.10×10^{-10} %. While seemingly minor, such errors can accumulate in large-scale computations, affecting final results. The use of the ysym() method enables symbolic arithmetic, emphasizing the importance of symbolic computation for accuracy. Although converting numeric to symbolic objects can be complex, especially with many variables, future research may explore alternative R packages to streamline this process. Moreover, this study is limited to linear systems with unique solutions and does not address non-linear systems or cases with multiple or no solutions. The scalability of symbolic computation to higher-dimensional matrices also remains a challenge due to increased computational overhead. These limitations highlight the need for further exploration into more efficient symbolic frameworks and broader applications in domains where accuracy is critical. As a direction for future research, we propose extending symbolic computation techniques to non-linear systems, exploring hybrid symbolic-numeric approaches, and evaluating performance across diverse application domains such as control systems, optimization, and machine learning.

Author Contributions

I Gusti Agung Anom Yudistira: Conceptualization, Methodology, Writing-Original Draft, Software, Validation, Writing-Reviewing and Editing. Kie Van Ivanky Saputra: Data Curation, Validation, Writing-Reviewing and Editing. All authors reviewed and approved the final manuscript.

Funding Statement

This research received no external funding.

Acknowledgment

The author would like to thank the R Core Team (https://www.r-project.org) and its contributors for continuously maintaining and developing the R system, enhancing its performance across versions while keeping it freely available. Gratitude is also extended to colleagues at Binus University and UPH for their support and the R training sessions, which have significantly contributed to the author's proficiency in using R. The author also wishes to express sincere appreciation to the reviewers and editors of *Barekeng: Jurnal Ilmu Matematika dan Terapan* for their valuable feedback and editorial guidance, which greatly improved the quality of this article.

Declarations

The authors declare that there are no competing interests or conflicts of interest related to this study.

REFERENCES

[1] H. Liu, S. Ling, L. Wang, Z. Yu, and X. Wang, "AN OPTIMIZED ALGORITHM AND THE VERIFICATION METHODS FOR IMPROVING THE VOLUMETRIC ERROR MODELING ACCURACY OF PRECISION MACHINE TOOLS," *The International Journal of Advanced Manufacturing Technology*, vol. 112, no. 11–12, pp. 3001–3015, Feb. 2021. doi: https://doi.org/10.1007/s00170-020-06266-x.

[2] J. M. Chambers, "OBJECT-ORIENTED PROGRAMMING, FUNCTIONAL PROGRAMMING AND R," *Statistical Science*, vol. 29, no. 2, pp. 167–180, 2014. doi: https://doi.org/10.1214/13-STS452.

[3] A. Z. Pinkus and S. Winitzki, "Yacas: A Do-It-Yourself Symbolic Algebra Environment."

[4] V. E. Dimiceli, A. S. I. D. Lang, and L. Locke, "TEACHING CALCULUS WITH WOLFRAM|ALPHA," *Int J Math Educ Sci Technol*, vol. 41, no. 8, pp. 1061–1071, Dec. 2010. doi: https://doi.org/10.1080/0020739X.2010.493241.

[5] E. Harahap, "PEMBELAJARAN ARITMATIKA MENGGUNAKAN APLIKASI WOLFRAM ALPHA (ARITHMETIC LEARNING USING WOLFRAM ALPHA APPLICATION)." [Online]. Available: https://journals.unisba.ac.id/index.php/Matematika

[6] P. Burns, *THE R INFERNO*, Second Edition. Lulu.com, 2012.

[7] F. de Dinechin, L. Forget, J.-M. Muller, and Y. Uguen, "POSITS," in *Proceedings of the Conference for Next Generation Arithmetic 2019*, New York, NY, USA: ACM, Mar. 2019, pp. 1–10. doi: https://doi.org/10.1145/3316279.3316285.

[8] H. Hernandez, "Rounding Error Propagation: Bias and Uncertainty," vol. 9, pp. 2024–2026. doi: https://doi.org/10.13140/RG.2.2.23005.59363.

[9] S. F. McCormick and R. Tamstorf, "ROUNDING-ERROR ANALYSIS OF MULTIGRID ({V})-CYCLES," *SIAM Journal on Scientific Computing*, vol. 46, no. 5, pp. S88–S95, Oct. 2024, doi: https://doi.org/10.1137/23M1582898.

[10] M. Marthaulina Lestari Siahaan, A. Rosa Da Lima Leli, K. Kefamenanu, and N. Tenggara Timur, "A STUDY OF LEARNING OBSTACLES: DETERMINING SOLUTIONS OF A SYSTEM OF LINEAR EQUATIONS USING GAUSS-JORDAN METHOD," Mosharafa: Jurnal Pendidikan Matematika, vol. 12, no. 1, 2023, [Online]. Available: http://journal.institutpendidikan.ac.id/index.php/mosharafa. doi: https://doi.org/10.31980/mosharafa.v12i1.1921

[11] I. G. A. A. Yudistira and R. Nariswari, "OPERASI DASAR BARIS/KOLOM MATRIKS SECARA INTERAKTIF DENGAN MENGGUNAKAN R," *Engineering, MAthematics and Computer Science (EMACS) Journal*, vol. 5, no. 1, pp. 25–32, Jan. 2023. doi: https://doi.org/10.21512/emacsjournal.v5i1.9206.

[12] S. Feng, Y. Feng, C. Yu, Y. Zhang, and H. X. Liu, "TESTING SCENARIO LIBRARY GENERATION FOR CONNECTED AND AUTOMATED VEHICLES, PART I: METHODOLOGY," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1573–1582, Mar. 2021. doi: https://doi.org/10.1109/TITS.2020.2972211.

[13] H. Edison, X. Wang, and K. Conboy, "COMPARING METHODS FOR LARGE-SCALE AGILE SOFTWARE DEVELOPMENT: A SYSTEMATIC LITERATURE REVIEW," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2709–2731, Aug. 2022. doi: https://doi.org/10.1109/TSE.2021.3069039.

[14] B. K. Das, D. N. Jha, S. K. Sahu, A. K. Yadav, R. K. Raman, and M. Kartikeyan, "INTRODUCTION TO R SOFTWARE," in *Concept Building in Fisheries Data Analysis*, Singapore: Springer Nature Singapore, 2023, pp. 209–233. doi: https://doi.org/10.1007/978-981-19-4411-6_12.

[15] M. M. Andersen and S. Højsgaard, "COMPUTER ALGEBRA IN R BRIDGES A GAP BETWEEN SYMBOLIC MATHEMATICS AND DATA IN THE TEACHING OF STATISTICS AND DATA SCIENCE." [Online]. Available: https://github.com/r-

[16] N. Kshetri, M. M. Rahman, O. F. Osama, and J. Hutson, "ALGOTRIC: SYMMETRIC AND ASYMMETRIC ENCRYPTION ALGORITHMS FOR CRYPTOGRAPHY-A COMPARATIVE ANALYSIS IN AI ERA." *Int. J. Adv. Comput. Sci. Appl. (IJACSA)*, vol. 15, no. 12, 2024

[17] S. A. Eisa and S. Pokhrel, "ANALYZING AND MIMICKING THE OPTIMIZED FLIGHT PHYSICS OF SOARING BIRDS: A DIFFERENTIAL GEOMETRIC CONTROL AND EXTREMUM SEEKING SYSTEM APPROACH WITH REAL TIME IMPLEMENTATION," *SIAM J Appl Math*, vol. 84, no. 3, pp. S82–S104, Jun. 2024. doi: https://doi.org/10.1137/22M1505566.

[18] M. Cui, J. Xu, Y. Zhou, H. Yang, L. Ji, and B. Zhou, "PESA: ERROR SENSITIVITY ANALYSIS TOOL FOR FLOATING-POINT COMPUTATIONAL PROGRAMS," *J Supercomput*, vol. 81, no. 3, p. 477, Feb. 2025. doi: https://doi.org/10.1007/s11227-025-06962-z.

[19] M. M. Andersen and S. Højsgaard, "COMPUTER ALGEBRA IN R WITH CARACAS," Apr. 2021, [Online]. Available: http://arxiv.org/abs/2104.05292. doi: https://doi.org/10.32614/CRAN.package.caracas

[20] Victor A. Bloomfield, *USING R FOR NUMERICAL ANALYSIS IN SCIENCE AND ENGINEERING*, 1st Edition. New York: Chapman and Hall/CRC, 2018.

[21] M. Andersen and S. Højsgaard, "RYACAS: A COMPUTER ALGEBRA SYSTEM IN R," *J Open Source Softw*, vol. 4, no. 42, p. 1763, Oct. 2019. doi: https://doi.org/10.21105/joss.01763.

[22] J. Divasón and J. Aransay, "GAUSS-JORDAN ALGORITHM AND ITS APPLICATIONS," 2016.

[23] A. T. Chantada, P. Protopapas, L. G. Bachar, S. J. Landau, and C. G. Scóccola, "EXACT AND APPROXIMATE ERROR BOUNDS FOR PHYSICS-INFORMED NEURAL NETWORKS," *arXiv preprint arXiv:2411.13848*, Nov. 2024.

[24] I Putu Alit Sudrastawa, "CONCEPTUAL AND PRACTICAL REVIEW OF GAUSSIAN ELIMINATION AND GAUSS-JORDAN REDUCTION," *JURNAL ILMU KOMPUTER INDONESIA*, vol. 7, no. 2, pp. 19–25, Nov. 2022.

[25] H. Anton, *ELEMENTARY LINEAR ALGEBRA*. John Wiley & Sons, Limited, 2018. [Online]. Available: https://books.google.co.id/books?id=ypROEAAAQBAJ

[26] S. Pal, K. Suresh, K. Suneetha, and M. B. Prabhakar, *LINEAR ALGEBRA*, 1st ed. Coimbatore, India: RK Publications, 2025.