Computational Gaussian Elimination

Jason Zhang

August 25, 2022

1 Gaussian Elimination

In linear algebra, Gaussian Elimination is an algorithm that consists of a sequence of operations performed on the coefficients of a matrix. In particular, Gaussian Elimination solves a system of linear equations. In general, given the matrix equation $Ax = b$, b can be solved by applying the Gaussian Elimination.

1.1 Row-Echelon Form

Definition: Row-Echelon Form (Reduced)

*A matrix is in **row-echelon form** if the following conditions are satisfied:*

1. *All **zero rows** (rows consisting entirely of zeros) are at the bottom of the matrix.*
2. *The leading coefficient **(pivot)** of a nonzero row is always right of the leading coefficient of the row above it.*
3. *The leading coefficient must have a value of 1 **(leading 1)**.*
   A matrix is in reduced row-echelon form with the additional condition:

*A matrix is in **reduced row-echelon form** with the additional condition:*

$\Rightarrow$ *Each leading 1 is the only nonzero entry in its column.*

The left matrix is in row-echelon form and right matrix is in *reduced* row echelon form:

$$\begin{bmatrix} 1 & * & * \\ 0 & 1 & * \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that a row-echelon matrix has an upper triangular matrix, where the asterisk denote arbitrary values.

In a reduced row-echelon matrix, if a column contains a leading 1, then there exist no other nonzero value in that column.

In both cases, the leading 1s possess a ”staircase” pattern.

Theorem: Row-Echelon Form Reduction

Any matrix can be brought to its row-echelon form and reduced row-echelon form through a sequence of elementary row operations. The sequence of elementary row operations is known as the Gaussian Elimination.

1.2 Gaussian Elimination

The sequence of elementary row operations is summarized in the Gaussian Elimination algorithm.

Theorem: Gaussian Elimination Algorithm

Given any matrix, the series of steps convert the original matrix into its reduced row-echelon form.
First, if the matrix is a zero matrix, then it is already in reduced row-echelon form.

1. *Starting from the first column from the left, find the row with the first nonzero entry (name it a). If the column contains no nonzero entry values, move onto the second column.*
2. *If a column with a nonzero entry is found, swap the row containing a with the first row.*
3. *Multiply the new top row by $\frac{1}{a}$ to create a leading 1.*
4. *Subtract multiples of that row from rows below it, such that each entry below the leading 1 is 0.*

The first iteration is completed.

In step 1, each further iteration must be conducted with one more column to the right of the former iteration column.
In step 2, the row containing a is swapped with one more row lower with each further iteration.

The algorithm is completed once all columns are iterated from step 1.

The code functionality will resemble the above algorithm formula.

1.3 Example

Below provides an example that demonstrates the implementation of the Gaussian Elimination algorithm.

Example: Gaussian Elimination

Given the matrix $A = \begin{bmatrix} 0 & 2 & -8 & 1 \\ 0 & 4 & 1 & 2 \\ 0 & 5 & 2 & 7 \\ 0 & 1 & 1 & 7 \end{bmatrix}$ *bring A into its reduced row-echelon form.*

Applying the Gaussian Elimination algorithm.

In step 1, starting with the first column, there are no nonzero entries; move onto the second column. In the second column's first row, there is a nonzero entry, value of 2.
In step 2, row swapping is not necessary.
In step 3, create a leading 1 by multiplying the first row by $\frac{1}{2}$, to yield the intermediate matrix:

$$\begin{bmatrix} 0 & 1 & -4 & \frac{1}{2} \\ 0 & 4 & 1 & 2 \\ 0 & 5 & 2 & 7 \\ 0 & 1 & 1 & 7 \end{bmatrix}$$

In step 4, add or subtract multiples of the top row from rows below it such that the entries below the leading 1 are 0.

Row 2: Subtracting 4 times row 1 from row 2
Row 3: Subtracting 5 times row 1 from row 3
Row 4: Subtracting 1 time row 1 from row 4

$$\begin{bmatrix} 0 & 1 & -4 & \frac{1}{2} \\ 0 & 0 & 17 & 0 \\ 0 & 0 & 22 & \frac{9}{2} \\ 0 & 0 & 5 & \frac{13}{2} \end{bmatrix}$$

Returning to step 1, starting with the third column and skipping to the second row, the first nonzero entry has a value of 17.
In step 2, row swapping is not necessary.
In step 3, create a leading 1 by multiplying the second row by $\frac{1}{17}$, to yield the intermediate matrix:

$$\begin{bmatrix} 0 & 1 & -4 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 22 & \frac{9}{2} \\ 0 & 0 & 5 & \frac{13}{2} \end{bmatrix}$$

In step 4, add or subtract multiples of the second row from rows above and below it such that the entries above and below the leading 1 are 0.

Row 1: Add 4 times row 2 from row 1
Row 3: Subtract 22 times row 2 from row 3
Row 4: Subtract 5 times row 2 from row 4

$$\begin{bmatrix} 0 & 1 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{9}{2} \\ 0 & 0 & 0 & \frac{13}{2} \end{bmatrix}$$

Returning to step 1, starting with the fourth column and skipping to the third row, the first nonzero entry has a value of $\frac{9}{2}$.
In step 2, row swapping is again not necessary.
In step 3, create a leading 1 by multiplying the third row by $\frac{2}{9}$, to yield the intermediate matrix:

$$\begin{bmatrix} 0 & 1 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{13}{2} \end{bmatrix}$$

In step 4, add or subtract multiples of the second row from rows above and below it such that the entries above and below the leading 1 are 0.

Row 1: Subtract $\frac{1}{2}$ times row 3 from row 1
Row 2: Already 0
Row 4: Subtract $\frac{13}{2}$ times row 3 from row 1

This final step yields the reduced row-echelon form of matrix A:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

2 Code Functionality

The program written in the C computer programming language functions the Gaussian Elimination algorithm. The complete code is linked here. The program prompts the matrix dimension, the number of rows and columns. Dynamic memory allocation is applied to store the matrix entries. The concerned block of code (written in C) is provided below:

```c
swapRowOfZeros(matrix, row, col);
int rows_counter = 0;
 int j;
 bool found;
 double saved_entry;

 for (int i = 0; i < col; i++){
 j = rows_counter;
 found = false;

      while(j < row && found == false){
          if (matrix[j][i] > 0.001 || matrix[j][i] < -0.001){
              if (rows_counter != j){
                  swapRows(matrix, rows_counter, j, col);
              }
              saved_entry = matrix[rows_counter][i];

              for (int i_col = 0; i_col < col; i_col++){
                  matrix[rows_counter][i_col] =
                  matrix[rows_counter][i_col] / saved_entry;
              }
              columnsToZero(matrix, row, col, rows_counter, i);
22                  rows_counter++;
23                  found = true;
24              }

25              else {
26                  j++;
27              }
28          }
29      }
```

In line 1, `swapRowofZeros` is a function that shifts rows of 0s of the original matrix to the bottom of the matrix.

2.1 Step 1

From the Gaussian Elimination Algorithm Theorem, step 1 iterates through each column and row of the matrix to find the first nonzero entry.

The first for loop (line 7) with index `i`, iterates each column of the matrix. The inner while loop (line 10) iterates each row of the matrix. As such, line 11 evaluates whether the iterated entry is a nonzero or zero entry.

2.2 Step 2

In step 2, the Gaussian Elimination Algorithm Theorem swaps the nonzero entry row with a respective row, namely `rows_counter`.

If a nonzero entry is found, line 12 first evaluates whether the rows to swap are the same. If the rows to swap are different, then line 13 applies the function `swapRows` to swap the row containing the nonzero entry, `j` with `rows_counter`.

2.3 Step 3

In step 3, the Gaussian Elimination Algorithm Theorem multiplies the row containing the nonzero entry by $\frac{1}{a}$ to create a leading 1.

The inner for loop (line 17) divides the concerning row by the reciprocal nonzero entry value.

2.4 Step 4

In step 4, the Gaussian Elimination Algorithm Theorem adds or subtracts multiples of the row containing the nonzero entry from rows above and below it. The multiple to apply is equal to the entry above and below the leading 1. This allows the entries above and below the leading 1 to equal 0.

Line 21 applies the function `columnsToZero` to make entries above and below the leading 1 zero. Lines 22 and 23 are incremented. `rows_counter` is incremented because the row to swap with the nonzero entry row is decreased by 1 each time a nonzero is found. `found` is equated to `true` because the first nonzero entry is found in the column; and thus the need to exit the while loop.

3 Computational Expense

The time complexity to conduct the Gaussian Elimination algorithm is $\mathcal{O}(n^3)$ where n is the number of rows. To explain, consider the worst case, where the number of rows and columns are equal. First, step 1 iterates through each column, and the computation expense is $\mathcal{O}(n)$. Second, the existence of the inner while loop iterates through each row, and the computation expense is again $\mathcal{O}(n)$. Last, the third nested for loop iterates through the columns of the matrix, and the computation expense is $\mathcal{O}(n)$. Amalgamating the computational expenses approximates the Gaussian Elimination algorithm to be $\mathcal{O}(n^3)$