Gaussian Elimination Code Explanation

Name: Sanghyeon Kim

Student ID: 20181605

Date: 09/21/2020

When Gaussian elimination is done by hand, the only thing to look out for is the human error such as calculation error. However, when gaussian elimination is done on a computer, one must consider that the computer does not hold rational numbers as quotients. Instead, the computer formally accepts rational numbers as a float and thus is limited in the number of decimal point numbers it can hold. Therefore, when performing a series of floating-point arithmetic, which is likely to happen in gaussian elimination, the computer may accumulate errors caused by the representation of rational numbers as floats with limited length. Therefore, an additional process called **partial pivoting** is used in order to minimize such errors.

Partial pivoting replaces the pivot row with another row that has a number in the pivot column with the largest absolute value below the pivot position. If the original pivot is the largest, the order is kept the same. The reason for this process can be explained intuitively. If the row with the largest number in the column becomes the pivot, the numbers in that row must be scaled down during the row operation. If errors were accumulated, the errors can be scaled down instead of being scaled up to perform row operation.

The program accepts the number of rows and columns of the augmented matrix as its first input. Then it asks the user to input the elements of the matrix for each row. The elements accepted must be an integer or a floating-point number. The program then creates an array of double pointers equal to the number of rows of the matrix, and each element in the array is assigned a double pointer to an allocated memory equaling the number of columns of the matrix. Pointers are used to make row exchange more convenient.

Then the program proceeds with the Gaussian elimination using partial pivoting. The program outputs a row-echelon form of the augmented matrix, which is the result of the Gaussian elimination.

```c
/*
Name:   gaussian_elimination.c

Author: Sanghyeon Kim(Student ID: 20181605)

Date:   09/21/2020

------------------------------------------------------------
Input:  Takes the size of an m*n-augmented matrix and each
        of the elements in the matrix from the user.


Process:Gaussian elimination is done with partial pivoting
        to cope with floating point accuracy.

Output: Prints the row-echelon form of the augmented
        matrix.

*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

void exchange_row(double **row1, double **row2);
void row_operation(double **M, int row_n, int col_n, int row_index, int col_in
dex, int col_max);
bool is_absMax(double prev_max, double new_num);

int main(){
    int row_n, col_n;
    int row_index, col_index;
    double col_max;

    // Get the number of rows and columns of the matrix as input
    printf("Type the number of rows and columns of the augmented matrix: ");
    scanf("%d %d", &row_n, &col_n);

    double *M[row_n];
    // Allocate contiguous memory for each row
    // Using a pointer variable comes in handy when performing a row exchange.
    for(int i = 0; i < row_n; ++i){
        *(M+i) = (double*) calloc(col_n, sizeof(double));
        if(!(M+i)){
            printf("calloc failed\n");
            return -1;
        }
    }
```

```
    // Get the row-
wise elements of the matrix as input and assign each element in M[row_i][col_j
]
    for(int i = 0; i < row_n; ++i){
        printf("Type the elements on row %d of the matrix: ", i+1);
        for(int j = 0; j < col_n; ++j){
            scanf("%lf", *(M+i)+j);
        }
    }

    row_index = 0;
    col_index = 0;

    // Performs a Gaussian elimination with partial pivoting
    while(col_index < col_n && row_index < row_n){
        col_max = M[row_index][col_index];
        // The loop finds the pivot with the largest absolute value in a colum
n
        for(int r = row_index+1; r < row_n; ++r){
            if(is_absMax(col_max, M[r][col_index])){
                // When a larger pivot is found exchange rows to put it in a p
ivot position
                col_max = M[r][col_index];
                exchange_row(M+row_index, M+r);
            }
        }
        if(col_max != 0){
            // If the column has at least one none zero value do the row opera
tion
            row_operation(M, row_n, col_n, row_index, col_index, col_max);
            row_index++;
        }
        // Move on to the next column
        col_index++;
    }

    printf("\n");

    //print the row echelon form of the matrix
    for(int i = 0; i < row_n; ++i){
        for(int j = 0; j < col_n; ++j){
            if(j == col_n-1){
                printf("| ");
            }
            printf("%lf ", *(*(M+i)+j));
        }
        printf("\n");
```

```
    }

    return 0;
}

// Exchange two rows of the matrix
void exchange_row(double **row1, double **row2){
    double *temp;

    temp = *row1;
    *row1 = *row2;
    *row2 = temp;

    return;
}

/*
    Returns true if current pivot is smaller than
    another number selected in the column and false
    otherwise.
*/
bool is_absMax(double curr_max, double new_num){
    if(curr_max < 0)
        curr_max = curr_max * -1;
    if(new_num < 0)
        new_num = new_num * -1;

    return curr_max < new_num;
}

/*
    Row operation is done in order to create zeros in the pivot column
    below the pivot.
*/
void row_operation(double **M, int row_n, int col_n, int pivot_row, int pivot_
col, int col_max){
    double pivot = M[pivot_row][pivot_col];
    for(int r = pivot_row+1; r < row_n; r++){
        double scale = M[r][pivot_col];
        // Subtract the scaled pivot row from the other rows
        for(int c = pivot_col; c < col_n; ++c){
            M[r][c] -= (scale/pivot * M[pivot_row][c]);
        }
    }
    return;
}
```