Gaussian elimination

Consider the following problem you will see in your course on linear algebra of

> Finding a row-equivalent matrix that is in row-echelon form given a matrix that represents a system of linear equations.

Gaussian elimination

Assuming that no pivoting is necessary, recall the Gaussian elimination algorithm that you learned (or will learn) in linear algebra. If we have an augmented matrix that has dimensions m x $(n + 1)$ (representing m linear equations in n unknowns), it may something like the following:

$$\left(\begin{array}{cccccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & a_{1,9} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} & a_{2,9} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} & a_{3,9} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & a_{4,9} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & a_{5,9} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & a_{6,9} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} \\ a_{8,1} & a_{8,2} & a_{8,3} & a_{8,4} & a_{8,5} & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} \end{array}\right)$$

This is an example of a matrix that represents a system of eight linear equations in eight unknowns. The goal of Gaussian elimination is to convert this into a row-equivalent matrix (i.e., one that has the same solution) which is of the form

$$\left(\begin{array}{cccccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & a_{1,9} \\ 0 & \tilde{a}_{2,2} & \tilde{a}_{2,3} & \tilde{a}_{2,4} & \tilde{a}_{2,5} & \tilde{a}_{2,6} & \tilde{a}_{2,7} & \tilde{a}_{2,8} & \tilde{a}_{2,9} \\ 0 & 0 & \tilde{a}_{3,3} & \tilde{a}_{3,4} & \tilde{a}_{3,5} & \tilde{a}_{3,6} & \tilde{a}_{3,7} & \tilde{a}_{3,8} & \tilde{a}_{3,9} \\ 0 & 0 & 0 & \tilde{a}_{4,4} & \tilde{a}_{4,5} & \tilde{a}_{4,6} & \tilde{a}_{4,7} & \tilde{a}_{4,8} & \tilde{a}_{4,9} \\ 0 & 0 & 0 & 0 & \tilde{a}_{5,5} & \tilde{a}_{5,6} & \tilde{a}_{5,7} & \tilde{a}_{5,8} & \tilde{a}_{5,9} \\ 0 & 0 & 0 & 0 & 0 & \tilde{a}_{6,6} & \tilde{a}_{6,7} & \tilde{a}_{6,8} & \tilde{a}_{6,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & \tilde{a}_{7,7} & \tilde{a}_{7,8} & \tilde{a}_{7,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \tilde{a}_{8,8} & \tilde{a}_{8,9} \end{array}\right)$$

We put little tilde symbols above most of the as because these values will not be the same as they are in the original augmented matrix. Once it is in this form, we may then use backward substitution to find the solution or solutions (assuming they exist).

The algorithm of Gaussian elimination proceeds as follows:

1. For each column starting at $j = 1, 2, \ldots, n + 1$:

a. For each row starting at $i = j + 1, \ldots, m$:
    i. Subtract an appropriate multiple of Row j onto Row i so as to eliminate the entry $a_{i,j}$ at location (i, j) in the matrx.

To begin, we will have two local variables that we could call m and n, but we will assume that these two variables are `num_equations` and `num_unknowns`, for numbers of equations and unknowns, respectively.

We can start with the outer loop:

```
for ( int j{1}; j <= num_unknowns + 1; ++j ) {
    // For each row going from j + 1 up to m,
    //    subtract an appropriate multiple of Row 'j'
    //    onto Row 'i'
}
```

Note that we are using `j` as a loop variable first and not i. This is because the entries of a matrix are usually represented as $a_{i,j}$ and the j here represents the column.

Thus, we must create our next loop, and there we will have a loop variable `i` which goes from `j + 1` to `num_equations`.

```
for ( int j{1}; j <= num_unknowns + 1; ++j ) {
    for ( int i{j + 1}; i <= num_equations; ++i ) {
        // Add an appropriate multiple of Row 'j'
        // onto Row 'i'
    }
}
```

Next, we must calculate the appropriate multiple of Row `j` to be subtracted from Row `i`. It happens that this coefficient is the negation of the $(i, j)^{\text{th}}$ entry of the augmented matrix divided by the $(j, j)^{\text{th}}$ entry.

```
for ( int j{1}; j <= num_unknowns + 1; ++j ) {
    for ( int i{j + 1}; i <= num_equations; ++i ) {
        // Calculate c = a     / a
        //               i,j /   j,j       i,j

        // Subtract 'c' times Row 'j' from Row 'i'.
    }
}
```

For example, if `j == 3` and `i == 7`, we would have an intermediate row-equivalent matrix that looks like the following:

$$\left(\begin{array}{cccccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & a_{1,9} \\ 0 & \tilde{a}_{2,2} & \tilde{a}_{2,3} & \tilde{a}_{2,4} & \tilde{a}_{2,5} & \tilde{a}_{2,6} & \tilde{a}_{2,7} & \tilde{a}_{2,8} & \tilde{a}_{2,9} \\ 0 & 0 & \tilde{a}_{3,3} & \tilde{a}_{3,4} & \tilde{a}_{3,5} & \tilde{a}_{3,6} & \tilde{a}_{3,7} & \tilde{a}_{3,8} & \tilde{a}_{3,9} \\ 0 & 0 & 0 & \tilde{a}_{4,4} & \tilde{a}_{4,5} & \tilde{a}_{4,6} & \tilde{a}_{4,7} & \tilde{a}_{4,8} & \tilde{a}_{4,9} \\ 0 & 0 & 0 & \tilde{a}_{5,4} & \tilde{a}_{5,5} & \tilde{a}_{5,6} & \tilde{a}_{5,7} & \tilde{a}_{5,8} & \tilde{a}_{5,9} \\ 0 & 0 & 0 & \tilde{a}_{6,4} & \tilde{a}_{6,5} & \tilde{a}_{6,6} & \tilde{a}_{6,7} & \tilde{a}_{6,8} & \tilde{a}_{6,9} \\ 0 & 0 & \tilde{a}_{7,3} & \tilde{a}_{7,4} & \tilde{a}_{7,5} & \tilde{a}_{7,6} & \tilde{a}_{7,7} & \tilde{a}_{7,8} & \tilde{a}_{7,9} \\ 0 & 0 & \tilde{a}_{8,3} & \tilde{a}_{8,4} & \tilde{a}_{8,5} & \tilde{a}_{8,6} & \tilde{a}_{8,7} & \tilde{a}_{8,8} & \tilde{a}_{8,9} \end{array}\right)$$

We would subtract $c = \dfrac{\tilde{a}_{7,3}}{\tilde{a}_{3,3}}$ times Row 3 from Row 7, and this would put a 0 at the entry (7, 3) of the matrix. To subtract one row of a matrix from another row, this requires a further loop, where we go through each of the entries of Row *i* and subtract from that entry the corresponding entry in Row *j*. A row has `num_unknowns + 1` entries, so this loop should run from `1` to this upper bound:

```
for ( int j{1}; j <= num_unknowns + 1; ++j ) {
    for ( int i{j + 1}; i <= num_equations; ++i ) {
        // Calculate c = a     / a
        //               i,j /   j,j

        for ( int k{1}; k <= num_unknowns + 1; ++k ) {
            // Subtract 'c' times a    from the entry a
            //                     j,k                i,k
        }
    }
}
```

Reducing our work

In the above matrix, when we subtract c times $\tilde{a}_{3,3}$ from $\tilde{a}_{7,3}$, we know what the result will be: zero. Thus, we are we doing the calculation we really don't need to, because we know what the result will be. You'll also notice that all the entries to the left of both Row `j` and Row `i` are zero, so we're doing a lot of work adding c times zero onto zero. Thus, we really don't have to do all this. We can simplify this as follows:

```
for ( int j{1}; j <= num_unknowns + 1; ++j ) {
    for ( int i{j + 1}; i <= num_equations; ++i ) {
        // Calculate c = a     / a
        //               i,j /   j,j

        // Assign a    = 0
        //         i,j

        for ( int k{j + 1}; k <= num_unknowns + 1; ++k ) {
            // Subtract 'c' times a    from the entry a
            //                    j,k                 i,j
        }
    }
}
```

Summary

In this example, we have not discussed how to do the various calculations, because that requires us to use a type called a two-dimensional array; a topic we will cover later. You do, however, know how to work with matrices, and therefore, we are discussing the algorithm in terms of the loops that are required.